Leibniz Transactions on Embedded Systems

Volume 4 | Issue 1 | February 2017

Vol. 4, Issue 1 ISSN 2199-2002 http://www.dagstuhl.de/lites

ISSN 2199-2002

Published online and open access by
the European Design and Automation Association (EDAA) / EMbedded Systems Special Interest Group (EMSIG) and Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
Online available at
http://www.dagstuhl.de/dagpub/2199-2002.

Publication date
February 2017

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier
10.4230/LITES-v004-i001

Aims and Scope
LITES aims at the publication of high-quality scholarly articles, ensuring efficient submission, reviewing, and publishing procedures. All articles are published open access, i.e., accessible online without any costs. The rights are retained by the author(s).

LITES publishes original articles on all aspects of embedded computer systems, in particular: the design, the implementation, the verification, and the testing of embedded hardware and software systems; the theoretical foundations; single-core, multi-processor, and networked architectures and their energy consumption and predictability properties; reliability and fault tolerance; security properties; and on applications in the avionics, the automotive, the telecommunication, the medical, and the production domains.

Contact
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
LITES, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
lites@dagstuhl.de
http://www.dagstuhl.de/lites

Contents

Regular Papers

A Note on the Period Enforcer Algorithm for Self-Suspending Tasks
Jian-Jia Chen and Björn B. Brandenburg 1:1–1:22

Utility-Based Scheduling of (m, k)-firm Real-Time Tasks – New Empirical Results
Florian Kluge 2:1–2:25

Special Issue on Quantitative Evaluation of Systems

Quantitative Analysis of Consistency in NoSQL Key-Value Stores
Si Liu, Jatin Ganhotra, Muntasir Raihan Rahman, Son Nguyen, Indranil Gupta, and José Meseguer 3:1–3:26

How Is Your Satellite Doing? Battery Kinetics with Recharging and Uncertainty
Holger Hermanns, Jan Krčál, and Gilles Nies 4:1–4:28

Characterizing Data Dependence Constraints for Dynamic Reliability Using n-Queens Attack Domains
Eric W. D. Rozier, Kristin Y. Rozier, and Ulya Bayram 5:1–5:26

A Note on the Period Enforcer Algorithm for Self-Suspending Tasks*

Jian-Jia Chen[1] and Björn B. Brandenburg[2]

1 **TU Dortmund University**
Dortmund, Germany
jian-jia.chen@cs.uni-dortmund.de

2 **Max Planck Institute for Software Systems (MPI-SWS)**
Kaiserslautern, Germany
http://orcid.org/0000-0001-8254-3815
bbb@mpi-sws.org

Abstract

The *period enforcer* algorithm for self-suspending real-time tasks is a technique for suppressing the "back-to-back" scheduling penalty associated with deferred execution. Originally proposed in 1991, the algorithm has attracted renewed interest in recent years. This note revisits the algorithm in the light of recent developments in the analysis of self-suspending tasks, carefully re-examines and explains its underlying assumptions and limitations, and points out three observations that have not been made in the literature to date: **(i)** period enforcement is not strictly superior (compared to the base case without enforcement) as it can cause deadline misses in self-suspending task sets that are schedulable without enforcement; **(ii)** to match the assumptions underlying the analysis of the period enforcer, a schedulability analysis of self-suspending tasks subject to period enforcement requires a task set transformation for which no solution is known in the general case, and which is subject to exponential time complexity (with current techniques) in the limited case of a single self-suspending task; and **(iii)** the period enforcer algorithm is incompatible with all existing analyses of suspension-based locking protocols, and can in fact cause ever-increasing suspension times until a deadline is missed.

2012 ACM Subject Classification Real-time systems, Real-time schedulability, Synchronization
Keywords and phrases period enforcer, deferred execution, self-suspension, blocking
Digital Object Identifier 10.4230/LITES-v004-i001-a001
Received 2015-12-15 **Accepted** 2016-08-18 **Published** 2016-12-21

1 Introduction

When real-time tasks suspend themselves (due to blocking I/O, lock contention, *etc.*), they defer a part of their execution to be processed at a later time. A consequence of such deferred execution is a potential interference penalty for lower-priority tasks [1, 12, 18, 17, 25, 26, 31]. This penalty, which is maximized when a task defers the completion of one job just until the release of the next job, can manifest as response-time increases and thus may lead to deadline misses.

To avoid such detrimental effects, Rajkumar [27] proposed the *period enforcer* algorithm, a technique to control (or shape) the processor demand of self-suspending tasks on uniprocessors and partitioned multiprocessors under preemptive fixed-priority scheduling. In a nutshell, the period enforcer algorithm artificially increases the length of certain suspensions whenever a task's activation pattern carries the risk of inducing undue interference in lower-priority tasks.

* This work has been supported by DFG, as part of the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Data Analysis".

The period enforcer algorithm is worth a second look for a number of reasons. First, in the words of Rajkumar, it "forces tasks to behave like ideal periodic tasks from the scheduling point of view with no associated scheduling penalties" [27], which is obviously highly desirable in many practical applications in which self-suspensions are inevitable (e.g., when offloading computations to co-processors such as GPUs or DSPs). Second, the later-proposed, but more widely-known *release guard* algorithm [32] uses a technique quite similar to period enforcement to control scheduling penalties due to release jitter in distributed systems. The period enforcer algorithm has also attracted renewed attention in recent years and has been discussed in several current works (e.g., [9, 11, 13, 15, 16, 14, 20, 21, 22, 23]), at times controversially [5]. And last but not least, the period enforcer algorithm plays a significant role in Rajkumar's seminal book on real-time synchronization [28].

In this note, we revisit the period enforcer [27] to carefully re-examine and explain its underlying assumptions and limitations, and to point out potential misconceptions. The main contributions are three observations that, to the best of our knowledge, have not been previously reported in the literature on real-time systems:

1. period enforcement can be a cause of deadline misses in self-suspending task sets that are otherwise schedulable (Section 3);
2. to match the assumptions underlying the analysis of the period enforcer, a schedulability analysis of self-suspending tasks subject to period enforcement requires a task set transformation for which no solution is known in the general case, and which is subject to exponential time complexity (with current techniques) in the limited case of a single self-suspending task (Section 4); and
3. the period enforcer algorithm is incompatible with all existing analyses of suspension-based locking protocols, and can in fact cause ever-increasing suspension times until a deadline is missed (Section 5).

We introduce the needed background in Section 2, restate our contributions more precisely in Section 2.4, and then establish the three above observations in detail in Sections 3–5 before concluding in Section 6.

2 Preliminaries

The period enforcer algorithm [27] applies to self-suspending tasks on uniprocessors under fixed-priority scheduling, and hence by extension also to multiprocessors under partitioned fixed-priority scheduling (where tasks are statically assigned to processors and each processor is scheduled as a uniprocessor). In this section, we review the underlying task model (Section 2.1), introduce the period enforcer algorithm (Section 2.2), summarize its analysis (Section 2.3), and finally restate our observations in more precise terms (Section 2.4).

2.1 Task Models

Since the analysis of the period enforcer requires reasoning about different task models and their relationships, we carefully introduce and precisely define the relevant models in this section.

2.1.1 Periodic Tasks

The most basic and best understood task model is the *periodic task model* due to Liu and Layland [19]. In this model, each task τ_i is characterized as a tuple (C_i, T_i), where C_i denotes an upper bound on the total execution time of any job of τ_i and T_i denotes the (exact) *inter-arrival time* (or *period*) of τ_i. Each such periodic task τ_i releases a job at time 0, and periodically every T_i

units thereafter. Each job must finish by the time the next arrives. Importantly, Liu and Layland assume both that the k^{th} job of τ_i arrives *exactly* at time $(k-1) \times T_i$, and that an incomplete job is *always* available for execution (i.e., jobs never block on I/O or locks).

A straightforward generalization of the periodic task model is to introduce an explicit *relative deadline* parameter D_i. In this case, each task is represented by a three-tuple (C_i, T_i, D_i), with the interpretation that every job of τ_i must finish within D_i time units after its arrival. Task τ_i is said to have an *implicit deadline* if $D_i = T_i$, a *constrained deadline* if $D_i \leq T_i$, and an *arbitrary deadline* otherwise. We primarily consider implicit deadlines in this note.

2.1.2 Sporadic Tasks

Mok [24] introduced the *sporadic task model*, a widely used generalization of the periodic task model in which each task τ_i is still specified by a tuple (C_i, T_i, D_i). However, the sporadic task model relaxes the inter-arrival constraint T_i to specify a *minimum* (rather than an exact) separation between jobs. In this interpretation, the first job does not necessarily arrive at time 0, and the exact arrival times of future jobs cannot be predicted, which is an appropriate modeling assumption for event-triggered tasks.

On uniprocessors, the relaxation from periodic to sporadic job arrivals does not introduce additional pessimism:[1] since any two jobs of a sporadic task τ_i are known to arrive *at least* T_i time units apart, the sporadic task model [24] still allows for schedulability analysis that is as accurate as Liu and Layland's analysis of periodic tasks [19].

Mok retained the assumption that incomplete jobs are always ready for execution (i.e., no suspensions), and that jobs are *immediately* available for execution after their arrival.

2.1.3 Release Jitter

The latter assumption – immediate availability for execution – is inappropriate in many practical systems (especially in networked systems) if events (e.g., messages) that trigger job releases can incur non-negligible delays (e.g., network congestion). Such delays in task activation can be accounted for by introducing a notion of *release jitter*. To this end, each task is represented by a four-tuple (C_i, J_i, T_i, D_i), where the parameter J_i is a bound on the maximum time that a job remains unavailable for execution after it should have started to run. Release jitter can be incorporated in both the periodic and the sporadic task models.

In the presence of release jitter, the terms "job arrival" and "job release," which are often used interchangeably, take on distinct meanings: a job's *release time* denotes the point in time when it actually becomes available for execution, whereas a job's *arrival time* is the instant that is relevant for the (minimum) inter-arrival time constraint. Any job of task τ_i is *released* at most J_i time units after it *arrives*.

Notably, non-zero release jitter *does* cause additional pessimism: in the worst case, two consecutive jobs of a task τ_i can be separated by as little as $T_i - J_i$ time units (if the earlier job incurs maximum release jitter and the successor job incurs none). As a result, a task may "carry in" some additional work into a given interval. Taking this effect into account, Audsley et al. [1] developed a response-time analysis for sporadic and periodic constrained-deadline tasks subject to release jitter under preemptive fixed-priority scheduling.

However, even in the presence of release jitter, a key assumption remains that jobs do not

[1] Assuming that all periodic tasks synchronously release a job at time zero.

self-suspend (e.g., wait for I/O).[2] That is, Audsley et al. [1] assume that, once a job is released, it continuously remains available for dispatching until it completes. This restriction is removed next.

2.1.4 Self-Suspending Tasks

When a job *self-suspends*, it becomes unavailable for execution until some external event occurs (e.g., a disk I/O operation completes, a network packet arrives, a co-processor signals completion, *etc.*). This has the effect of *deferring* (a part of) the job's processing requirement until the time that it *resumes* from its suspension, which causes massive analytical difficulties [1, 10, 12, 18, 17, 25, 26, 27, 30, 31].

To date, the real-time literature on self-suspensions has focused on two task models: the *dynamic* self-suspension model, which we discuss first, and the (*multi-*)*segmented* suspension model, which we discuss next in Section 2.1.5. Self-suspensions can arise in both periodic and sporadic tasks (i.e., both interpretations of the T_i parameter are possible). The observations that we make in this note apply equally to both periodic and sporadic tasks; for convenience, we focus primarily on periodic tasks.

The dynamic self-suspending task model characterizes each task τ_i as a four-tuple (C_i, S_i, T_i, D_i): the parameters C_i, T_i, and D_i have their usual meaning (i.e., as in the periodic and sporadic task models), and S_i denotes an upper bound on the total self-suspension time of any job of τ_i. The dynamic self-suspension model does not impose a bound on the maximum number of self-suspensions, nor does it make any assumptions as to where during a job's execution self-suspensions occur. That is, how often a job defers its execution, when it does so, and how much of its execution it defers may vary unpredictably from job to job.

Allowing tasks to self-suspend can impose substantial scheduling penalties (an example is provided shortly in Section 2.2) and greatly complicates schedulability analysis (e.g., see [25, 30, 10]). In particular, release jitter and self-suspensions are not interchangeable concepts and it is not safe [10, 25] to simply substitute J_i with S_i in Audsley et al.'s analysis [1]. (Nonetheless, under the dynamic suspension model, it is possible for jobs of self-suspending tasks to defer their entire execution requirement, so self-suspension can be seen as a generalization of release jitter.)

The period enforcer algorithm aims to mitigate the negative effects of self-suspensions. However, for reasons that will be explained in Section 2.2.4, the period enforcer algorithm cannot be meaningfully combined with the dynamic suspension model. Instead, it requires the segmented suspension model, which we discuss next.

2.1.5 Segmented Self-Suspending Tasks

The (multi-)segmented self-suspending sporadic task model extends the four-tuple (C_i, S_i, T_i, D_i) by characterizing each self-suspending task as a fixed, finite linear sequence of computation and suspension intervals. These intervals are represented as a tuple $(S_i^0, C_i^1, S_i^1, C_i^2, S_i^2, \ldots, S_i^{m_i-1}, C_i^{m_i})$, which is composed of m_i computation segments separated by m_i suspension intervals.

The first self-suspension segment S_i^0, prior to the first execution segment, is equivalent to release jitter (i.e., the parameter J_i in Section 2.1.3). However, in much of the literature on the segmented self-suspending task model, the segment S_i^0 is assumed to be absent (i.e., $S_i^0 = 0$), such that there are only $m_i - 1$ suspension intervals (and jobs arrive jitter-free). We adopt this convention and assume $S_i^0 = 0$ unless noted otherwise.

[2] Audsley et al. [1] do present a response-time analysis that takes into account a limited form of suspensions due to semaphores ("blocking"). However, their analysis does not apply to general self-suspensions (i.e., the kind of self-suspensions targeted by the period enforcer algorithm) and is not relevant in the context of this paper.

We say that a segment is *released* when it becomes available for execution. The first computation segment is released immediately when the job arrives (unless $S_i^0 \neq 0$); the second computation segment (if any) is released when the job resumes from its first self-suspension, *etc.*

The advantage of the dynamic model (Section 2.1.4) is that it is more flexible since it does not impose any assumptions on a task's control flow. The advantage of the segmented model is that it allows for more accurate analysis. The period enforcer algorithm and its analysis [27] applies (only) to the segmented model, as explained in Sections 2.2.4 and 2.3.

A note on terminology: for the sake of consistency with the recent literature on self-suspensions in real-time systems, we favor the term "segmented self-suspending tasks" to refer to tasks under the just-introduced model. However, Rajkumar's original description of the period enforcer [27] refers to such tasks as *deferrable tasks*, as it predates the widespread adoption of the former term. We use both terms interchangeably in this paper.

2.1.6 Single-Segment Self-Suspending (aka Deferrable) Tasks

An important special case is segmented self-suspending tasks with exactly one self-suspension interval followed by exactly one computation segment ($m_i = 1$, $S_i^0 \neq 0$), which we refer to as *single-segment self-suspending tasks*. This special case is central to Rajkumar's original analysis of the period enforcer [27], as we will explain in Section 2.3. Regarding terminology, Rajkumar [27] does not use a special term for single-segment self-suspending tasks, simply referring to them as deferrable tasks. To avoid ambiguity, we instead explicitly mention the "single-segment" qualifier.

Note also that single-segment self-suspending sporadic tasks, which are "suspended" only prior to commencing execution, are analytically fully equivalent to sporadic tasks subject to release jitter (i.e., the model described in Section 2.1.3). We nonetheless use the term "single-segment self-suspending task," or interchangeably "single-segment deferrable task," to remain close to Rajkumar's original description [27], and to highlight the connection to the (multi-)segmented self-suspending task model (Section 2.1.5).

This concludes our review of relevant task models. Before reviewing the period enforcer and its original analysis, we briefly introduce some essential concepts.

2.1.7 Assumptions, Busy Intervals, and Task Set Transformations

We focus exclusively on preemptive fixed-priority scheduling in this note, as the period enforcer is explicitly designed for this setting. For simplicity, we assume that tasks are indexed in order of decreasing priority (i.e., τ_1 is the highest-priority task).

A key concept in the period enforcer's runtime rules (discussed next) is the notion of a *level-i busy interval*, which is a maximal interval during which the processor executes only segments of tasks with priority i or higher.

Finally, Rajkumar's original analysis [27] of the period enforcer is rooted in the concept of a *task set transformation*. In general, such a task set transformation is simply a function f that maps a given task set $\mathcal{T}$ to a transformed task set $\mathcal{T}' = f(\mathcal{T})$ *such that $\mathcal{T}'$ is schedulable* ***only if*** *the original task set $\mathcal{T}$ is schedulable*, too. The basic idea is that such a transformation allows schedulability analysis by reduction: given a suitable transformation f, $\mathcal{T}$ can be *indirectly* shown to be schedulable by computing $\mathcal{T}' = f(\mathcal{T})$ and establishing that $\mathcal{T}'$ is schedulable.

Importantly, the tasks in $\mathcal{T}$ and $\mathcal{T}'$ do *not* have to be of the same task model, nor does the number of tasks have to remain the same (i.e., $|\mathcal{T}| \neq |\mathcal{T}'|$ is possible). Specifically, the task set transformation underlying the analysis of the period enforcer maps each *multi*-segmented self-suspending task $\tau_i \in \mathcal{T}$ to m_i *single*-segmented self-suspending tasks in $\mathcal{T}'$ (i.e., $|\mathcal{T}'| = \sum_{\mathcal{T}} m_i$).

With these definitions in place, we can now introduce the period enforcer.

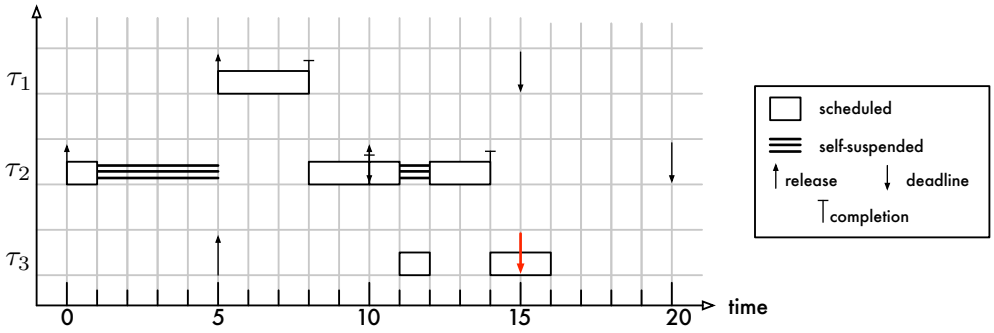


Figure 1 Example uniprocessor schedule (*without* period enforcement) of three tasks τ_1, τ_2, and τ_3 with periods $T_1 = T_2 = T_3 = 10$. Tasks τ_1 and τ_3 consist of a single computation segment ($C_1^1 = C_3^1 = 3$); task τ_2 consists of two computation and one suspension segment ($C_2^1 = 1$, $S_2^1 = 4$, $C_2^2 = 2$). Jobs of tasks τ_1 and τ_3 are released just as τ_2 resumes from its self-suspension at time 5. Without period enforcement, task τ_3 misses a deadline at time 15 because the second job of task τ_2 suspends only briefly (for one time unit rather than four).

2.2 The Period Enforcer Algorithm

The period enforcer consists of two parts: a runtime rule that governs when each segment of a self-suspending task may be scheduled, and an (offline) analysis that may be used to assess the temporal correctness of a set of self-suspending tasks (Section 2.1.5) subject to period enforcement. Initially, we focus on the runtime rule (i.e., the actual period enforcer algorithm) and then review the corresponding original analysis thereafter in Section 2.3. We begin with a simple example that highlights the effect that the period enforcer is designed to control.

2.2.1 The Problem: Back-to-Back Execution

The scheduling penalty associated with self-suspensions is maximized when a task defers the completion of one job just until the release of the next job. This effect is illustrated in Figure 1, which shows a case in which the self-suspension of the higher-priority task τ_2 from time 1 until time 5 results in a deadline miss of the lower-priority task τ_3 at time 15.

The root cause is increased interference due to the "back-to-back" execution effect [1, 18, 17, 26, 31]. In the example shown in Figure 1, two jobs of τ_2 execute in close succession (i.e., separated by less than a period) because the second job, released at time 10, self-suspended for a (much) shorter duration than the first job. Consequently, τ_3 suffers from increased interference when τ_2's second job resumes "too soon" at time 12 after having been suspended for only one time unit, rather than four time units like the first job of τ_2.

2.2.2 The Period Enforcement Rule

The key idea underlying the period enforcer algorithm is to artificially delay the execution of computation segments if a job resumes "too soon." To this end, the period enforcer determines for each computation segment an *eligibility time*. If a segment resumes before its eligibility time, the execution of the segment is delayed until the eligibility time is reached.

A segment's eligibility time is determined according to the following rule. Let $ET_{i,j}^k$ denote the eligibility time of the k^{th} computation segment of the j^{th} job of task τ_i. Further, let $a_{i,j}^k$ denote the segment's arrival time. Finally, let $busy(\tau_i, t')$ denote the last time that a level-i busy interval begins on or prior to time t' (i.e., the processor executes only τ_i or higher-priority tasks throughout

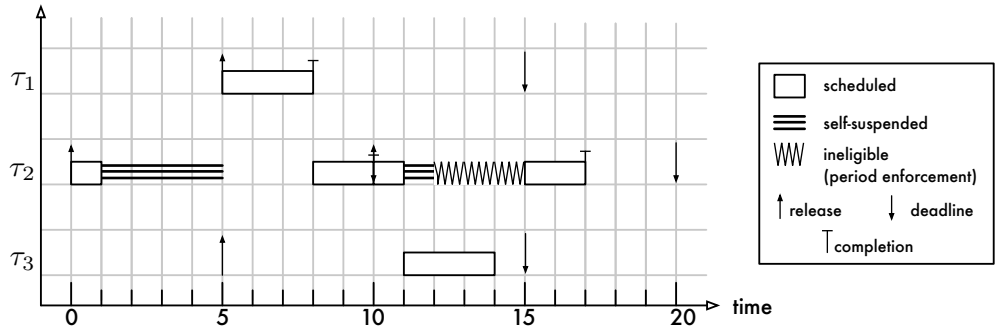


Figure 2 Example uniprocessor schedule *with* period enforcement assuming the same scenario as depicted in Figure 1. With period enforcement, task τ_3 does not miss a deadline because task τ_2's second computation segment is delayed until time 15 when it no longer imposes undue interference (i.e., it is prevented from resuming "too soon" at time 12).

the interval $[busy(\tau_i, t'), t'])$. The period enforcer algorithm defines the segment eligibility time of the k^{th} segment as

$$ET^k_{i,j} = \max\left(ET^k_{i,j-1} + T_i,\ busy(\tau_i, a^k_{i,j})\right), \tag{1}$$

where $ET^k_{i,0} = -T_i$ [27, Section 3.1]. This simple and elegant rule has the desirable effect of avoiding all back-to-back execution, which can be easily observed with an example.

2.2.3 Example: Avoiding Back-to-Back Execution

Figure 2 illustrates how the definition of eligibility time in Equation (1) restores the schedulability of the task set depicted in Figure 1. Consider the eligibility times of the second segment of task τ_2.

By definition, $ET^2_{2,0} = -T_2 = -10$. At time 5, when the second computation segment of the first job resumes ($a^2_{2,1} = 5$), we thus have

$$ET^2_{2,1} = \max\left(-T_2 + T_2,\ busy(\tau_2, a^2_{2,1})\right) = \max(0, 5) = 5$$

since the release of τ_2's second segment (and the arrival of τ_1) starts a new level-2 busy interval at time $a^2_{2,1} = 5$. The second segment of τ_2's first job is hence immediately eligible to execute; however, due to the presence of a pending higher-priority job, τ_2 is not actually scheduled until time 8 (just as without period enforcement as depicted in Figure 1).

The second segment of the second job of τ_2 is released at time $a^2_{2,2} = 12$. In this case, the segment is *not* immediately eligible to execute since

$$ET^2_{2,2} = \max\left(ET^2_{2,1} + T_2,\ busy(\tau_2, a^2_{2,2})\right) = \max(5 + 10, 12) = 15.$$

Hence, the execution of τ_2's second computation segment does not start until time $ET^2_{2,2} = 15$, which gives τ_3 sufficient time to finish before its deadline at time 15.

The examples in Figures 1 and 2 suggest an intuition for the benefits provided by period enforcement: computation segments of a self-suspending task τ_i are forced to execute at least T_i time units apart (hence the name), which ensures that it causes no more interference than a regular (non-self-suspending) sporadic task.

2.2.4 Incompatibility with the Dynamic Self-Suspension Model

Before reviewing the classic analysis based on this intuition, we briefly comment on the difficulty of combining period enforcement with the dynamic self-suspension model (Section 2.1.4).

In short, to be effective, the period enforcer fundamentally requires the segmented self-suspension model (Section 2.1.5) because it cannot cope with the unpredictable execution times between (the unpredictably many) self-suspensions that jobs may exhibit under the dynamic self-suspension model.

A simple example can explain why the period enforcer algorithm is not compatible with the dynamic self-suspending task model. Consider a trivial system that has only one task with a total execution time $C_1 = 1$, a total self-suspension length $S_1 = 1$, and a period and relative deadline of $D_1 = T_1 = 2$. Suppose the first job of task τ_1 arrives at time 0, suspends itself for one time unit, and then executes for one time unit. Further suppose the second job of task τ_1 arrives at time 2, first executes for 0.5 time units, then suspends for 1 time unit, and finally executes for 0.5 time units. With the period enforcer algorithm in place, the second job of task τ_1 starts its execution at time 3, at which point it will clearly miss its deadline at time 4.

In this example, the problem is that the eligibility time of the first computation "segment" of the second job is determined by the self-suspension pattern of the first job, even though the first job deferred all of its execution, whereas the second job deferred only a part of its execution. Under the more restrictive segmented self-suspension model (Section 2.1.5), the pattern of self-suspension and computation times is statically fixed; such a mismatch is hence not possible.

Next, we revisit the original analysis of the period enforcer algorithm.

2.3 Classic Analysis of the Period Enforcer Algorithm

The central notation in Rajkumar's analysis [27] is a deferrable task, which matches our notion of segmented tasks, as already discussed in Section 2.1.5. Specifically, Rajkumar states that:

> "With deferred execution, a task τ_i can execute its C_i units of execution in discrete amounts $C_i^1, C_i^2, \ldots$ with suspension in between C_i^j and C_i^{j+1}." [27, Section 3][3]

Central to Rajkumar's analysis [27] is a task set transformation (recall Section 2.1.7) that splits each deferrable task with multiple segments (Section 2.1.5) into a corresponding number of single-segment deferrable tasks (Section 2.1.6). In the words of Rajkumar [27, Section 3]:

> "Without any loss of generality, we shall assume that a task τ_i can defer its entire execution time but not parts of it. That is, a task τ_i executes for C_i units with no suspensions once it begins execution. Any task that does suspend after it executes for a while can be considered to be two or more tasks each with its own worst-case execution time. The only difference is that if a task τ_i is split into two tasks τ_i' followed by τ_i'', then τ_i'' has the same deadlines as τ_i'."

In other words, the transformation can be understood as splitting each self-suspending task into a matching number of single-segment deferrable tasks (Section 2.1.6), which are equivalent to non-self-suspending sporadic tasks subject to release jitter (Section 2.1.3), which can be easily analyzed with classic fixed-priority response-time analysis [1]. To constitute an effective schedulability analysis, the transformation must ensure that, if the transformed set of single-segment deferrable tasks can be shown to be schedulable (e.g., with response-time analysis [1]), then the original set of multi-segment deferrable tasks is also schedulable under period enforcement.

To summarize, as illustrated in Figure 1, uncontrolled deferred execution can impose increased interference on lower-priority tasks because of the potential for "back-to-back" execution [1, 18, 17, 26, 31]. The purpose of the period enforcer algorithm is to reduce such penalties for lower-priority

[3] The notation has been altered here for the sake of consistency.

tasks without detrimentally affecting the schedulability of self-suspending, higher-priority tasks. The latter aspect – no detrimental effects for self-suspending tasks – is captured concisely by Theorem 5 in the original analysis of the period enforcer algorithm [27].

> **Theorem 5**: A [single-segment] deferrable task that is schedulable under its worst-case conditions is also schedulable under the period enforcer algorithm [27].

The "worst-case conditions" mentioned in the theorem simply correspond to the case when **(i)** a job of a single-segment deferrable task defers its execution for the maximally allowed time S_i^0 (i.e., when it incurs maximal release jitter) and **(ii)** it incurs maximum higher-priority interference (i.e., when its start of execution coincides with a critical instant [19]).

2.4 Questions Answered in This Paper

Theorem 5 (in [27]) is a strong result: it implies that the period enforcer does not induce any deadline misses. This seemingly enables a powerful analysis approach: if the corresponding transformed set of single-segment deferrable tasks can be shown to be schedulable *without* period enforcement under fixed-priority scheduling using *any* applicable analysis (e.g., [1]), then the period enforcer algorithm also yields a correct schedule.

However, recall that, in the original analysis [27], deferrable tasks are assumed to defer their execution either completely or not at all (but not parts of it). It is hence important to realize that Theorem 5 in [27] applies only to the transformed set of *single-segment* deferrable tasks, and that it does *not* apply to the *original* set of multi-segmented self-suspending tasks.

This leads to the first question: *If the original set of segmented self-suspending tasks is schedulable without period enforcement, is it then also schedulable under period enforcement?* That is, can Theorem 5 (in [27]) be generalized to multi-segmented self-suspending tasks? In Section 3, we answer this question in the negative.

1. There exist sets of segmented self-suspending tasks that are schedulable under fixed-priority scheduling without any enforcement, but that are infeasible under period enforcement. This shows that Theorem 5 in [27] has to be used with care – it may be applied only in the context of the transformed single-segment deferrable task set, but not in the context of the original multi-segmented self-suspending task set.

Therefore, to apply Theorem 5 to conclude that a set of segmented self-suspending task sets remains schedulable despite period enforcement, we first have to answer the task-set transformation question: *given a set of segmented self-suspending tasks $\mathcal{T}$, how do we obtain a corresponding set of single-segment deferrable tasks $\mathcal{T}'$ such that $\mathcal{T}'$ is schedulable (without period enforcement) only if $\mathcal{T}$ is schedulable (with period enforcement)?* That is, as discussed in Section 2.3, the classic analysis of the period enforcer [27] presumes that it is possible to convert a set of multi-segmented self-suspending tasks into a corresponding set of single-segment deferrable tasks, but it is left undefined in [27] *how* this central step should be accomplished. In Section 4, we make a pertinent observation.

2. How to derive a single-segment deferrable task set corresponding to a given set of multi-segmented self-suspending tasks is an open problem. Recent findings by Nelissen et al. [25] can be applied in a special case, but their method takes exponential time (even in the special case).

Finally, we consider the use of the period enforcer in conjunction with suspension-based multiprocessor locking protocols for partitioned fixed-priority scheduling (such as the MPCP [15, 26] or the FMLP [2, 6]). While it is certainly tempting to apply period enforcement with the intention

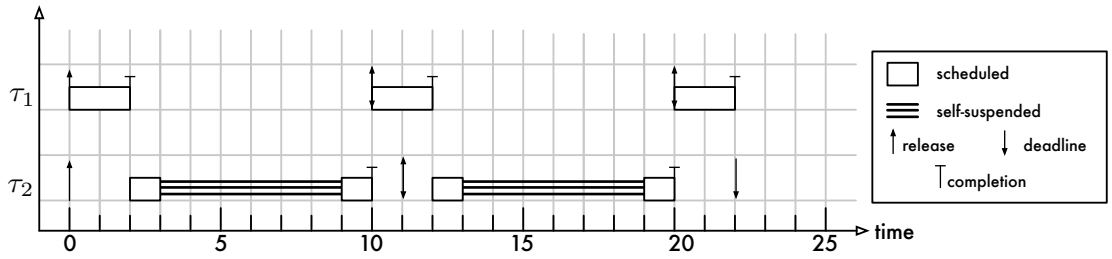


Figure 3 An illustrative example of the original self-suspending task set (without period enforcement) assuming periodic job arrivals on a uniprocessor. Task τ_1 has higher priority than task τ_2.

of avoiding the negative effects of deferred execution due to lock contention (as previously suggested elsewhere [15, 14, 28]), we ask: *does existing blocking analysis remain safe when combined with the period enforcer algorithm?* In Section 5, we show that this is not the case.

3. The period enforcer algorithm invalidates all existing blocking analyses for real-time semaphore protocols as there exist non-trivial feedback cycles between the period enforcer rules and blocking durations.

3 Period Enforcement Can Induce Deadline Misses

In this section, we demonstrate with an example that there exist sets of sporadic segmented self-suspending tasks that both **(i)** are schedulable *without* period enforcement and **(ii)** are not schedulable with period enforcement.

To this end, consider a task system consisting of 2 tasks. Let τ_1 denote a sporadic task without self-suspensions and parameters $C_1 = 2$ and $T_1 = D_1 = 10$, and let τ_2 denote a self-suspending task consisting of two segments with parameters $C_2^1 = 1$, $S_2^1 = 6$, $C_2^2 = 1$, and $T_2 = D_2 = 11$. Suppose that we use the rate-monotonic priority assignment, i.e., τ_1 has higher priority than τ_2. This task set is schedulable without any enforcement since at most one computation segment of a job of τ_2 can be delayed by τ_1:

- if the first segment of a job of τ_2 is interfered with by τ_1, then the second segment resumes at most 9 time units after the arrival of the job and the response time of task τ_2 is hence 10; otherwise,
- if the first segment of a job of τ_2 is not interfered with by τ_1, then the second segment resumes at most 7 time units after the arrival of the job and hence the response time of task τ_2 is at most 10 even if the second segment is interfered with by τ_1.

Figure 3 depicts an example schedule of the task set assuming periodic job arrivals.

Next, let us consider the same task set under control of the period enforcer algorithm, as defined in Section 2.2. Figure 4 shows the resulting schedule for a periodic release pattern. The first job of task τ_2 (which arrives at time $a_{2,1}^1 = 0$) is executed as if there is no period enforcement since the definition $ET_{2,0}^1 = ET_{2,0}^2 = -T_2$ ensures that both segments are immediately eligible. Note that the first segment of τ_2's first job is delayed due to interference from τ_1. As a result, the second segment of τ_2's first job does not resume until time $a_{2,1}^2 = 9$. Thus, we have

$$ET_{2,1}^1 = \max\left(-T_2 + T_2,\ busy(\tau_2, 0)\right) = 0 \text{ and}$$
$$ET_{2,1}^2 = \max\left(-T_2 + T_2,\ busy(\tau_2, 9)\right) = 9.$$

In contrast to the first job, the second job of task τ_2 (which arrives at time 11) is affected by period enforcement. The first segment of the second job is released at time $a_{2,2}^1 = 11$, incurs

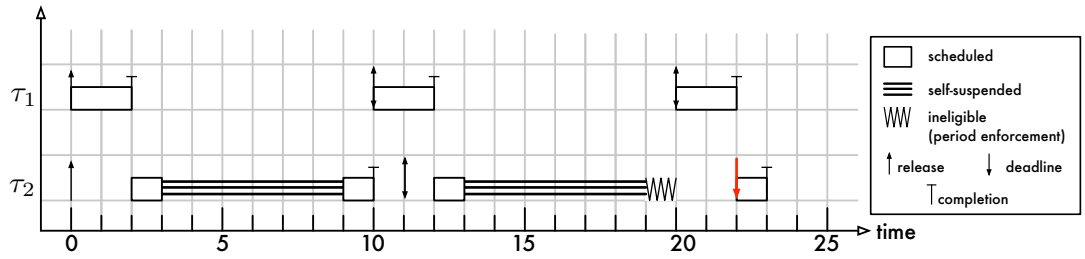


Figure 4 An illustrative example demonstrating a deadline miss at time 22 under the period enforcer algorithm. At time 19, τ_2 resumes, but it remains ineligible to execute until time 20 when τ_1 is released.

interference for one time unit during $[11, 12)$, and suspends at time 13. The second segment of the second job hence resumes only at time $a^2_{2,2} = 19$. Thus, we have

$$ET^1_{2,2} = \max\left(0 + 11,\ busy(\tau_2, 11)\right) = 11 \text{ and}$$
$$ET^2_{2,2} = \max\left(9 + 11,\ busy(\tau_2, 19)\right) = 20.$$

According to the rules of the period enforcer algorithm, the processor therefore remains idle at time 19 because the segment is not eligible to execute until time $ET^2_{2,2} = 20$. However, at time 20, the third job of τ_1 is released. As a result, the second job of τ_2 suffers from additional interference and misses its deadline at time 22.

This example shows that there exist sporadic segmented self-suspending task sets that **(i)** are schedulable under fixed-priority scheduling without any enforcement, but **(ii)** are not schedulable under the period enforcer algorithm.

One may consider to enrich the period enforcer with the following scheduling rule: when the processor becomes idle, a task immediately becomes eligible to execute regardless of its eligibility time. However, even with this extension, the above example remains valid by introducing one additional lower-priority task τ_3 with execution time $C_3 = 13$ (to be executed from time 3 to time 9 and time 13 to time 20) and $T_3 = D_3 = 100$. With task τ_3, the processor is always busy from time 0 to time 23 and consequently τ_2 still misses its deadline at time 22.

Furthermore, the example also demonstrates that the conversion to single-segment deferrable tasks does incur a loss of generality since it introduces pessimism. In the context of the above example, if we convert the multi-segmented suspending task τ_2 into two single-segment deferrable tasks, called τ^1_2 and τ^2_2, where task τ^1_2 never defers its execution and task τ^2_2 defers its execution by at most 9 time units, the resulting single-segment deferrable task set $\{\tau_1, \tau^1_2, \tau^2_2\}$ is in fact not schedulable under the given priority assignment: if a job of τ_1 coincides with the arrival of a job of τ^2_2 after it has maximally deferred its execution, the job of τ^2_2 has a response time of $9 + 2 + 1$ time units, which exceeds its relative deadline of 11 time units. This shows that any restriction to single-segment deferrable tasks — that is, assuming that "[w]ithout any loss of generality [...] a task τ_i can defer its entire execution time but not parts of it" [27] (recall Section 2.3) — does in fact come with a loss of generality.

4 Deriving a Corresponding Deferrable Task Set

To apply an analysis of the period enforcer based on Theorem 5 in [27], we first need to convert a given set of multi-segment self-suspending tasks into a corresponding set of single-segment deferrable tasks. This raises the question: how can we efficiently derive the corresponding set of single-segment deferrable tasks?

The original period enforcer proposal [27] is silent on this issue and does not spell out a procedure for converting a multi-segmented self-suspending task to a corresponding set of single-segment deferrable tasks. However, in our opinion, performing such a transformation without introducing additional pessimism is not at all easy in the general case.

In the following, we illustrate the inherent difficulty of the problem by focusing on a special case to which we can apply a recent result of Nelissen et al. [25], which allows analyzing the worst-case response time of multi-segmented self-suspending sporadic tasks, albeit with exponential time complexity. Specifically, Nelissen et al.'s worst-case response time analysis [25], which is based on *mixed-integer linear programming* (MILP), can be applied under the following conditions:

1. the task set contains only one self-suspending task,
2. the self-suspending task is the lowest-priority task,
3. the scheduling policy is preemptive fixed-priority scheduling, and
4. all tasks have constrained deadlines (i.e., $D_i \leq T_i$ for all τ_i).

As an aside, it is interesting to note that, even for the restrictive case above, Nelissen et al.'s MILP-based analysis [25] provides only an upper bound on the worst-case response time, and not necessarily the *exact* worst-case response time. However, at least conceptually, an exact answer *can* be obtained based on their analysis by combining it with an exhaustive search. Specifically, Lemma 2 of Nelissen et al. [25] provides a characterization of the worst-case release pattern that yields a maximal response time. Thus, by exploring all release patterns that satisfy the conditions stated in Lemma 2 of Nelissen et al. [25], an exact bound can be determined (at great computational cost). A detailed discussion of this approach can be found in a recent report [8].

Let us now return to the discussion of the task-set transformation that is needed before Theorem 5 in [27] can be applied. For an arbitrary number of tasks $k \geq 2$, suppose that the system has $k-1$ regular sporadic tasks and only one segmented self-suspending task τ_k, and that all tasks have implicit deadlines (i.e., $D_i = T_i$ for all τ_i). Further suppose that task τ_k has m_k segments with $m_k \geq 3$.

To convert a computation segment of τ_k into a single-segment deferrable task, we need to derive the segment's *latest-possible release time*, relative to the arrival of the corresponding job. Formally, for the j^{th} computation segment of task τ_k, we let ρ_k^j denote its latest-possible release time, with the interpretation that, if a job of task τ_k arrives at time t, then it is guaranteed that the j^{th} computation segment of this job will be released at the latest at time $t + \rho_k^j$.

How can we compute ρ_k^j? Suppose that the worst-case response time of the j^{th} computation segment of task τ_k is W_k^j, and recall that S_k^j denotes the maximum self-suspension length before the j^{th} computation segment of τ_k. Then ρ_k^j can be expressed in terms of W_k^{j-1}:

$$\rho_k^j = W_k^{j-1} + S_k^{j-1},$$

where $W_k^0 = 0$. Therefore, if we can derive the exact segment worst-case response time W_k^j for $j = 1, 2, \ldots, m_k - 1$, we can easily compute ρ_k^j for $j = 1, 2, \ldots, m_k$. And conversely, if we can somehow obtain ρ_k^j for $j = 2, \ldots, m_k$, we can trivially infer W_k^j for $j = 1, 2, \ldots, m_k - 1$. Based on these considerations, it appears that the transformation problem is – at least in the considered special case – equivalent to the worst-case response time analysis of a multi-segmented self-suspending task.

However, deriving an exact bound W_k^j for task τ_k and for $j = 1, 2, \ldots, m_k - 1$ is not easy: even for the above "simple" case, Nelissen et al.'s MILP solution [25] for calculating a safe upper bound on the worst-case response time requires exponential time complexity if $j \geq 2$. In fact, it has recently been shown that the problem of verifying the schedulability of such a task set is coNP-hard in the strong sense [7]. It follows analogously that analyzing the exact worst-case response time of task τ_k is NP-hard in the strong sense [7].

Notably, Nelissen et al. [25] and Chen [7] do not consider the period enforcer; rather, their results pertain to unrestricted self-suspensions. However, given that the period enforcer has no effect on tasks that do not self-suspend [27], and given that in the considered special case only the lowest-priority task self-suspends, we believe that their observations transfer to the period enforcement case.

To summarize, to analyze the period enforcer based on Theorem 5 in [27], a procedure for transforming multi-segmented self-suspending tasks into sets of single-segment deferrable tasks is needed, but no such procedure is given in the original proposal [27]. Based on the presented considerations, we conclude that filling in this missing step is non-trivial and observe that the closest known solution by Nelissen et al. [25] requires exponential time even in the greatly simplified special case of a single self-suspending task, and that the problem of verifying the schedulability of such a task set is in fact coNP-hard in the strong sense [7]. It thus remains unclear how Theorem 5 in [27] can be used for schedulability analysis of sets of multi-segmented self-suspending tasks. While we did search for alternative analysis approaches that do not rely on Theorem 5, we did not find a simple or efficient schedulability test for the period enforcer without introducing substantial additional pessimism. The problem remains open.

Next, we take a look at the period enforcer in the context of synchronization protocols.

5 Incompatibility with Suspension-Based Locking Protocols

Binary semaphores, i.e., suspension-based locks used to realize mutually exclusive access to shared resources, are a common source of self-suspensions in multiprocessor real-time systems. When a task tries to use a resource that has already been locked, it self-suspends until the resource becomes available. Such self-suspensions due to lock contention, just like any other self-suspension, result in deferred execution and thus can detrimentally affect a task's interference on lower-priority tasks. It may thus seem natural to apply the period enforcer to control the negative effects of blocking-induced self-suspensions.[4] However, as we demonstrate with two examples, it is actually unsafe to apply period enforcement to lock-induced self-suspensions.

5.1 Combining Period Enforcement and Suspension-Based Locks

Whenever a task attempts to lock a shared resource, it may potentially block and self-suspend. In the context of the multi-segmented self-suspending task model, each lock request hence marks the beginning of a new segment.

The period enforcer algorithm may therefore be applied to determine the eligibility time of each such segment (which, again, all start with a critical section). There is, however, one complication: when does a task actually *acquire* a lock? That is, if a task's execution is postponed due to the period enforcement rules, at which point is the lock request processed, with the consequence that the resource becomes unavailable to other tasks?

There are two possible interpretations of how period enforcement and locking rules may interact. Under the **first interpretation**, when a task requires a shared resource, which implies the beginning of a new segment, its lock request is processed *only when its new segment is eligible for execution*, as determined by the period enforcer algorithm. Alternatively, under the **second interpretation**, a task's request is processed *immediately* when it requires a shared resource.

[4] The use of period enforcement in combination with suspension-based locks has indeed been assumed in prior work [28], stated as a motivation and possible use case in the original period enforcer proposal [27], and suggested as a potential improvement elsewhere [15, 14].

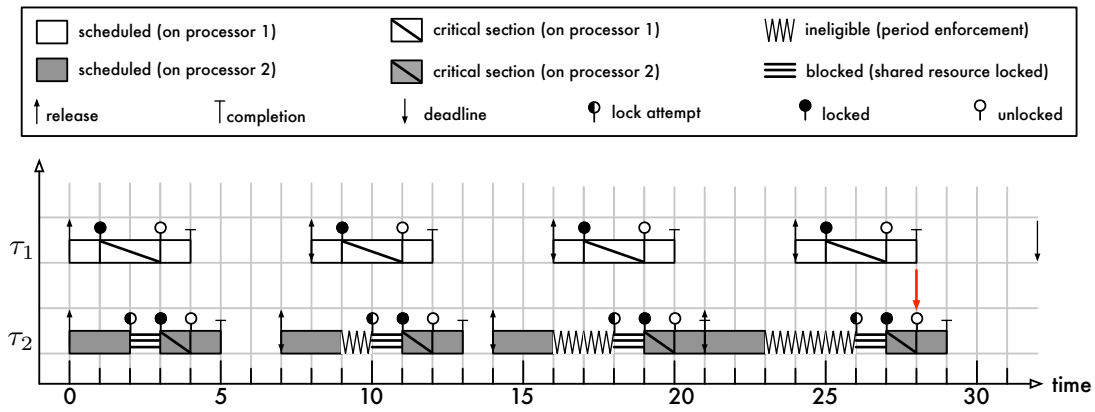


Figure 5 Example schedule of two tasks τ_1 and τ_2 on two processors sharing one lock-protected resource. The example assumes that lock requests take effect only when the critical section segment becomes eligible to be scheduled according to the rules of the period enforcer algorithm. Under this interpretation, the fourth job of task τ_2 misses its deadline at time 28.

As a consequence of the first rule, a task may find a required shared resource unavailable when its new segment becomes eligible for execution even though the resource was available when the prior segment finished. As a consequence of the second rule, a shared resource may be locked by a task that cannot currently use the resource because the task is still ineligible to execute.

We believe that the first interpretation is the more natural one, as it does not make much sense to allocate resources to tasks that cannot yet use them. However, for the sake of completeness, we show that either interpretation can lead to deadline misses even if the task set is trivially schedulable without any enforcement.

5.2 Case 1: Locking Takes Effect at Earliest Segment Eligibility Time

In the following example, we assume the first interpretation, i.e., that the processing of lock requests is delayed until the point when a resuming segment would no longer be subject to any delay due to period enforcement. We show that this interpretation leads to a deadline miss in a task set that would otherwise be trivially schedulable.

Consider the following simple task set consisting of two tasks on two processors that share one resource. Task τ_1, on processor 1, has a total execution cost of $C_1 = 4$ and a period and deadline of $T_1 = D_1 = 8$. After one time unit of execution, jobs of τ_1 require the shared resource for two time units. τ_1 thus consists of two segments with costs $C_1^1 = 1$ and $C_1^2 = 3$. Task τ_2, on processor 2, has the same overall WCET ($C_2 = 4$), a slightly shorter period ($T_2 = D_2 = 7$), and requires the shared resource for one time unit after *two* time units of execution ($C_2^1 = 2$ and $C_2^2 = 2$). Without period enforcement (and under any reasonable locking protocol), the task set is trivially schedulable because, by construction, any job of τ_1 incurs at most one time unit of blocking, and any job of τ_2 incurs at most two time units of blocking.

In contrast, with period enforcement, deadline misses are possible. Figure 5 depicts a schedule of the two tasks assuming periodic job arrivals and use of the period enforcer algorithm. We focus on the eligibility times $ET_{2,1}^2, ET_{2,2}^2, ET_{2,3}^2, \ldots$ of the second segment of τ_2.

Since τ_2's first job requests the shared resource only after two time units of execution, it is blocked by τ_1's critical section, which commenced at time 1. At time 3, τ_1 releases the shared resource and τ_2 consequently resumes (i.e., $a_{2,1}^2 = 3$). According to the period enforcer rules [27],

the second segment is immediately eligible because, according to Equation 1 (in Section 3),

$$ET^2_{2,1} = \max\left(ET^2_{2,0} + T_2,\ busy(\tau_2, a^2_{2,1})\right) = \max(-T_2 + T_2,\ 3) = 3.$$

(Recall that $ET^2_{2,0} = -T_2$, and interpret $busy(\tau_2, a^2_{2,1})$ with respect to τ_2's processor.)

At time 7, the second job of τ_2 is released. Its first segment ends at time 9. However, its second segment is not eligible to be scheduled before time 10 since $ET^2_{2,2} \geq ET^2_{2,1} + T_2 = 3 + 7 = 10$. At time 9, the second job of τ_1, released at time 8, can thus lock the shared resource without contention. Consequently, when τ_2's request for the shared resource takes effect at time 10, the resource is no longer available and τ_2 must wait until time $a^2_{2,2} = 11$ before it can proceed to execute. We thus have

$$ET^2_{2,2} = \max\left(ET^2_{2,1} + T_2,\ busy(\tau_2, a^2_{2,2})\right) = \max(10,\ 11) = 11.$$

The third job of τ_2 is released at time 14. Its first segment ends at time 16, but since $ET^2_{2,3} \geq ET^2_{2,2} + T_2 = 11 + 7 = 18$, the second segment may not commence execution until time 18 and the shared resource remains available to other tasks in the meantime. The third job of τ_1 is released at time 16 and acquires the uncontested shared resource at time 17. Thus, the segment of τ_2 cannot resume execution before time $a^2_{2,3} = 19$. Therefore

$$ET^2_{2,3} = \max\left(ET^2_{2,2} + T_2,\ busy(\tau_2, a^2_{2,3})\right) = \max(18,\ 19) = 19.$$

The same pattern repeats for the fourth job of τ_2, released at time 21: when its first segment ends at time 23, the second segment is not eligible to commence execution before time 26 since $ET^2_{2,4} \geq ET^2_{2,3} + T_2 = 19 + 7 = 26$. By then, however, τ_1 has already locked the shared semaphore again, and the second segment of the fourth job of τ_2 cannot resume before time $a^2_{2,4} = 27$, at which point

$$ET^2_{2,4} = \max\left(ET^2_{2,3} + T_2,\ busy(\tau_2, a^2_{2,4})\right) = \max(26,\ 27) = 27.$$

However, this leaves insufficient time to meet the job's deadline: as the second segment of τ_2 requires $C^2_2 = 2$ time units to complete, the job's deadline at time 28 is missed.

By construction, this example does not depend on a specific locking protocol; for instance, the effect occurs with both the MPCP [26] (based on priority queues) and the FMLP [2, 6] (based on FIFO queues). The corresponding response-time analyses for both protocols [3, 15] predict a worst-case response time of 6 for task τ_2 (i.e., four time units of execution, and at most two time units of blocking due to the critical section of τ_1). This demonstrates that, under the first interpretation, adding period enforcement to suspension-based locks invalidates existing blocking analyses. Furthermore, it is clear that the devised repeating pattern can be used to construct schedules in which the response time of τ_2 grows beyond any given implicit or constrained deadline.

Next, we show that the second interpretation can also lead to deadline misses in otherwise trivially schedulable task sets.

5.3 Case 2: Locking Takes Effect Immediately

From now on, we assume the second interpretation: all lock requests are processed immediately when they are made, even if this causes the shared resource to be locked by a task that is not yet eligible to execute according to the rules of the period enforcer algorithm. We construct an example in which a task's response time grows with each job until a deadline is missed.

To this end, consider two tasks with identical parameters hosted on two processors. Task τ_1 is hosted on processor 1; task τ_2 is hosted on processor 2. Both tasks have the same period and

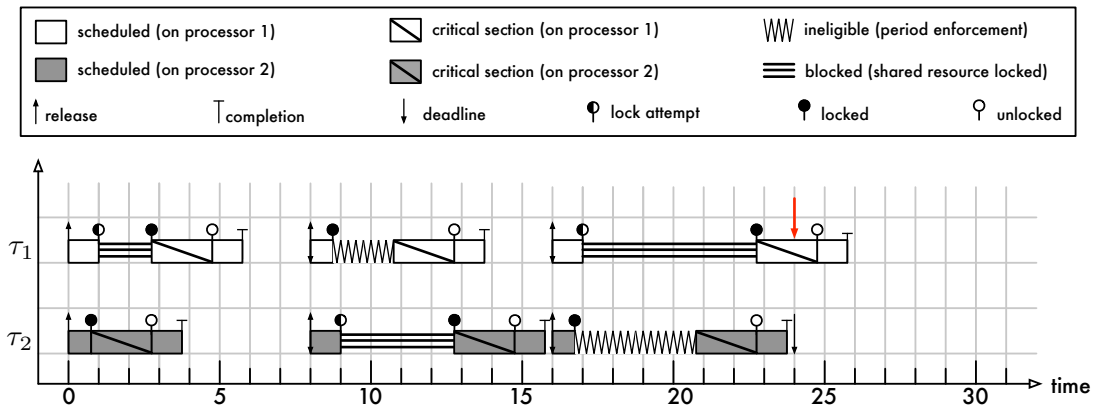


Figure 6 Example schedule of two tasks τ_1 and τ_2 on two processors sharing one lock-protected resource. The example assumes that lock requests take effect immediately, even if the critical section segment is not yet eligible to be scheduled according to the rules of the period enforcer algorithm. Under this interpretation, the third job of task τ_1 misses its deadline at time 24.

relative deadline $T_1 = T_2 = D_1 = D_2 = 8$ and the same WCET of $C_1 = C_2 = 4$. They both access a single shared resource for two time units each per job. Both tasks request the shared resource after executing for *at most* one time unit. They both thus have two segments each with parameters $C_1^1 = C_2^1 = 1$ and $C_1^2 = C_2^2 = 3$.

The example exploits that a job may require *less* service than its task's specified WCET. To ensure that the shared resource is acquired in a certain order, we assume the following deterministic pattern of the actual execution times. Let ϵ be an arbitrarily small, positive real number with $\epsilon < 1$.

- The first segment of even-numbered jobs of τ_1 executes for only $1 - \epsilon$ time units.
- The first segment of odd-numbered jobs of τ_2 executes for only $1 - \epsilon$ time units.
- All other segments execute for their specified worst-case costs.

Figure 6 shows an example schedule assuming periodic job arrivals.

At time $1 - \epsilon$, the first job of τ_2 acquires the shared resource because τ_1 does not issue its request until time 1. Consequently, τ_1 is blocked until time $a_{1,1}^2 = 3 - \epsilon$, and we have

$$ET_{1,1}^2 = \max\left(ET_{1,0}^2 + T_1,\ busy(\tau_1, a_{1,1}^2)\right) = \max(-T_1 + T_1,\ 3 - \epsilon) = 3 - \epsilon$$

and

$$ET_{2,1}^2 = \max\left(ET_{2,0}^2 + T_2,\ busy(\tau_2, a_{2,1}^2)\right) = \max(-T_2 + T_2,\ 0) = 0.$$

The roles of the second jobs of both tasks are reversed: since the second job of τ_1 locks the shared resource already at time $9 - \epsilon$, τ_2 is blocked when it attempts to lock the resource at time 9. However, according to the rules of the period enforcer algorithm, the second segment of the second job of τ_1 is not actually eligible to execute before time $11 - \epsilon$ since

$$ET_{1,2}^2 = \max\left(ET_{1,1}^2 + T_1,\ busy(\tau_1, a_{1,2}^2)\right) = \max(3 - \epsilon + 8,\ 8) = 11 - \epsilon.$$

Consequently, even though the lock is granted to τ_1 already at time $9 - \epsilon$, the critical section is executed only starting at time $11 - \epsilon$, and τ_2 is thus delayed until time $13 - \epsilon$. At time $13 - \epsilon$, τ_2 is immediately eligible to execute since

$$ET_{2,2}^2 = \max\left(ET_{2,1}^2 + T_2,\ busy(\tau_2, a_{2,2}^2)\right) = \max(0 + 8,\ 13 - \epsilon) = 13 - \epsilon.$$

The third jobs of both tasks are released at time 16. The roles are swapped again: because τ_2's first segment requires only $1 - \epsilon$ time units of service, it acquires the lock at time $a^2_{2,3} = 17 - \epsilon$, before τ_1 issues its request at time 17. However, according to the period enforcer algorithm's eligibility criterium, τ_2 cannot actually continue its execution before time $21 - \epsilon$ since

$$ET^2_{2,3} = \max\left(ET^2_{2,2} + T_2,\ busy(\tau_2, a^2_{2,3})\right) = \max(13 - \epsilon + 8,\ 16) = 21 - \epsilon.$$

This, however, means that τ_1 cannot use the shared resource before time $23 - \epsilon$, which leaves insufficient time to complete the second segment of τ_1's third job before its deadline at time 24. Furthermore, if both tasks continue the illustrated execution pattern, the period enforcer continues to increase their response times. As a result, the pattern may be repeated to construct schedules in which any arbitrarily large implicit or constrained deadline is violated.

As in the previous example, the response-time analyses for both the MPCP [3, 15] and the FMLP [3] predict a worst-case response time of 6 for both tasks (i.e., four time units of execution, and at most two time units of blocking). The example thus demonstrates that, if lock requests take effect immediately, then the period enforcer is incompatible with existing blocking analyses because, under the second interpretation, it increases the effective lock-holding times.

5.4 Other Protocols and Interpretations

The examples in Sections 5.2 and 5.3 assume a *shared-memory* locking protocol: once a lock is granted, tasks execute their own critical sections on their assigned processors. One may wonder whether effects similar to those described in Sections 5.2 and 5.3 can also occur under *distributed* real-time locking protocols such as the *Distributed Priority Ceiling Protocol* (DPCP) [28, 29] or the *Distributed FIFO Locking Protocol* (DFLP) [3, 4], where critical sections may be executed on dedicated *synchronization processors*. In this case, the self-suspension occurs on the task's *application processor*, which is different from the (remote) synchronization processor on which the critical section is executed.

This separation allows employing period enforcement only on application processors (while avoiding it on synchronization processors) without incurring the feedback cycle between blocking times and self-suspension times highlighted in Sections 5.2 and 5.3.

However, period enforcement still invalidates all existing blocking analyses for distributed real-time semaphore protocols [3, 28, 29] because it artificially increases blocking times if tasks contain multiple accesses to shared resources. An example demonstrating this effect is shown in Figure 7. Two segmented self-suspending tasks τ_1 and τ_2 share a resource using a distributed real-time locking protocol. The choice of protocol is irrelevant; the example works with both the DPCP and the DFLP. The tasks have parameters $m_1 = 2$, $C^1_1 = C^2_1 = 1$, $T_1 = D_1 = 25$ and $m_2 = 3$, $C^1_2 = C^3_2 = 1$, $C^2_2 = 6$, and $T_2 = D_2 = 15$. The computation segments are separated by self-suspensions that arise while the tasks wait for the completion of critical sections that are executed remotely on a dedicated synchronization processor P_{sync}; the corresponding suspension segment parameters S^1_1, S^1_2, and S^2_2 will be defined shortly.

The first jobs of τ_1 and τ_2 are both released at time 0 and attempt to access the shared resource at time 1. Task τ_1's request is serviced first; as a result τ_2 resumes only at time $a^2_{2,1} = ET^2_{2,1} = 5$ after having been suspended for four time units:

$$ET^2_{2,1} = \max\left(ET^2_{2,0} + T_2,\ busy(\tau_2, a^2_{2,1})\right) = \max(-T_2 + T_2,\ 5) = 5.$$

Task τ_2 then executes its second computation segment for $C^2_2 = 6$ time units until time 11, when the job accesses the shared resource for a second time. Since there is no contention from τ_1 at this time, τ_2 resumes after only two time units at time 13. This leaves the job sufficient time to complete at time 14, one time unit before its deadline at time 15.

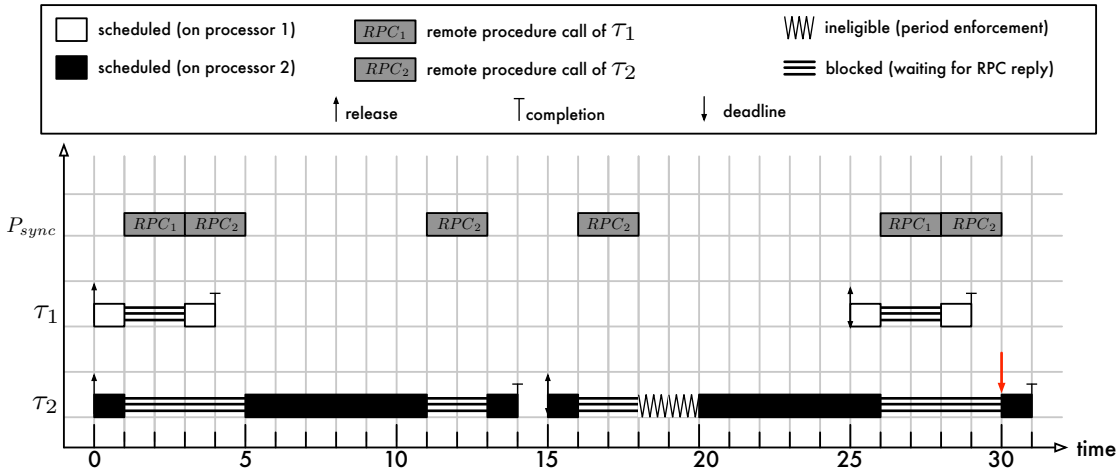


Figure 7 Example schedule of two tasks τ_1 and τ_2 on two processors sharing a *remote* resource using a distributed semaphore protocol (e.g., the DPCP [28, 29] or the DFLP [3, 4]) together with the period enforcer. Since critical sections are executed as *remote procedure calls* (RPCs) on a dedicated synchronization processor P_{sync} (not subject to period enforcement), their execution is not delayed by period enforcement. However, period enforcement delays τ_2 at time 18, which invalidates existing analyses [3, 28, 29]: under both the DPCP and the DFLP, τ_2 is predicted to have a maximum response time of 14 [3], but with period enforcement, the second job of τ_2 has in fact a response time of 16.

The second job of τ_2 is released at time 15 and issues a request for the shared resource at time 16. Since there is no contention from τ_1 at the time, the second computation segment arrives already at time $a^2_{2,2} = 18$, after having been self-suspended for only two time units. However, since the second segment of the first job arrived at time $a^2_{2,1} = 5$, the second segment of the second job is not eligible to start execution until time $ET^2_{2,2} = 20$ since

$$ET^2_{2,2} = \max\left(ET^2_{2,1} + T_2,\ busy(\tau_2, a^2_{2,2})\right) = \max(5 + T_2,\ 18) = 20.$$

As a result, τ_2 faces contention from τ_1 when it issues its second request for the shared resource at time 26, which ultimately leads to a deadline miss at time 30.

In contrast, without period enforcement, τ_2 does not miss its deadline at time 30 because, *across its two requests*, a job of τ_2 is delayed by at most one request of τ_1, for a total self-suspension time of at most two six time units. That is, even though the *individual* self-suspension segments of the two tasks are each up to four time units long (i.e., $S^1_1 = S^1_2 = S^2_2 = 4$), the fact that self-suspensions arise due to the same cause (resource contention) means that the *total* self-suspension time is actually less than the sum of the individual per-segment bounds.

Existing analyses for the DPCP [3, 28, 29] and the DFLP [3] exploit this knowledge and therefore predict task τ_2 to be schedulable with a worst-case response time of 14.[5] The example in Figure 7 thus demonstrates that the period enforcer invalidates existing blocking bounds for distributed semaphore protocols. As an aside, the example in Figure 7 further highlights limitations of the segmented self-suspension model in the context of synchronization protocols, where the lengths of self-suspensions encountered at runtime are inherently not independent.

[5] The analyses in [28, 29] do assume a segmented task model, but bound the total blocking across all segments. The analysis in [3] also bounds the total blocking across all segments and can be applied to both the segmented and the dynamic self-suspension model.

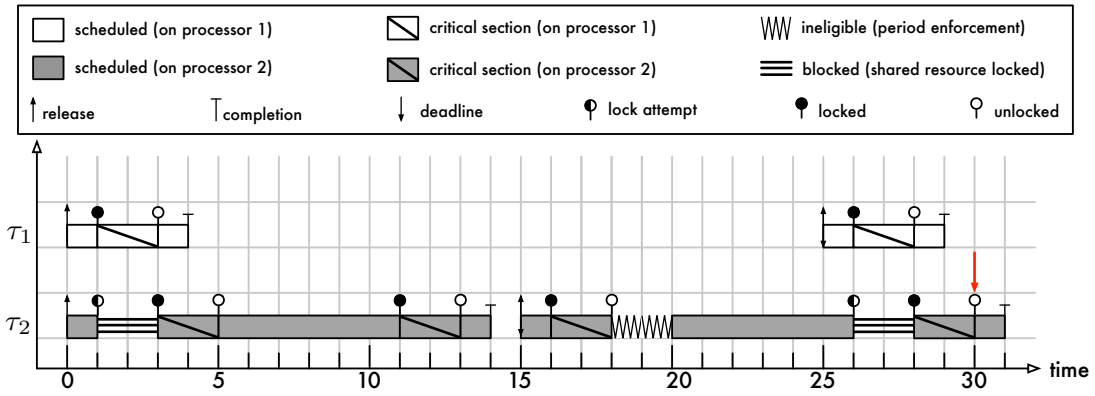


Figure 8 Example schedule of two tasks τ_1 and τ_2 on two processors sharing one lock-protected resource. The example assumes that lock requests take effect immediately and that critical sections are exempt from period enforcement (i.e., period enforcement is applied only to any computation after a critical section). As in Figure 7, the second job of task τ_2 misses its deadline at time 30.

Returning to the shared-memory case, as a third possible interpretation, one could also exclude critical sections from period enforcement such that only the rest of the computation segment *after* a critical section is subject to period enforcement (i.e., making critical sections immediately eligible to execute).[6] This can be understood as making each critical section an individual computation segment (exempt from period enforcement) that is separated from the following computation by a "virtual" self-suspension of maximum length zero. As in the case of distributed semaphore protocols, this interpretation breaks the feedback cycle highlighted in Sections 5.2 and 5.3, but still invalidates all existing blocking analyses as it artificially inflates the synchronization delay.

An example of this effect is shown in Figure 8, which depicts the same scenario as in Figure 7 under the assumption that a shared-memory semaphore protocol is used (i.e., critical sections are executed locally by each job) and that critical sections are exempt from period enforcement. As in the distributed case, period enforcement induces a deadline miss, whereas existing blocking analyses [3, 15] exploit the fact that a remote critical section can block only once, thus arriving at a worst-case response time bound of 14 for τ_2.

To conclude, in both Figures 7 and 8, period enforcement has an effect *as if* a single remote critical section can block a given job multiple times, which is fundamentally incompatible with efficient blocking analysis [3].

5.5 Discussion

While it is intuitively appealing to combine period enforcement with suspension-based locking protocols [15, 14, 28], we observe that this causes non-trivial difficulties. In particular, our examples show that the addition of period enforcement invalidates all existing blocking analyses.

If critical sections are subject to period enforcement, our examples also suggest that devising a correct blocking analysis would be a substantial challenge due to the demonstrated feedback cycle between the period enforcer rules and blocking durations. Fundamentally, the design of the period enforcer algorithm implicitly rests on the assumption that a segment *can* execute as soon as it is eligible to do so. In the presence of locks, however, this assumption is invalidated. As demonstrated, the result can be a successive growth of self-suspension times that proceeds until a

[6] This interpretation does not fit the assumptions stated in [27, 28].

deadline is missed. The period enforcer algorithm, at least as defined and used in the literature to date [27, 28], is therefore incompatible with the existing literature on suspension-based real-time locking protocols (e.g., [2, 3, 15, 14, 28]).

Finally, it is worth noting that our examples can be trivially extended with lower-priority tasks to ensure that no processor idles before the described deadline misses occur. It is also not difficult to extend the examples in Figures 6 and 8 with a task on a third processor such that all critical sections of τ_1 and τ_2 are separated from their predecessor segments by a non-zero self-suspension.

6 Concluding Remarks

We have revisited the underlying assumptions and limitations of the period enforcer algorithm, which Rajkumar [27] introduced to handle segmented self-suspending real-time tasks.

One key assumption in the original proposal [27] is that a deferrable task τ_i can defer its entire execution time but not parts of it. This creates some mismatches between the original segmented self-suspending task set and the corresponding single-segment deferrable task set, which we have demonstrated with an example that shows that Theorem 5 in [27] does not reflect the schedulability of the original segmented self-suspending task system.

The original proposal [27] further left open the question of how to convert a segmented self-suspending task set to a corresponding set of single-segment deferrable tasks. This problem remains open. Taking into account recent developments [7, 8, 25], we have observed that such a transformation is non-trivial in the general case.

Finally, we have demonstrated that substantial difficulties arise if one attempts to combine suspension-based locks with period enforcement. These difficulties stem from the fact that period enforcement can increase contention or lock-holding times, which increases the lengths of self-suspension intervals, which then in turn feeds back into the period enforcer's minimum suspension lengths. As a consequence, period enforcement invalidates all existing blocking analyses.

Nevertheless, the period enforcer algorithm *per se*, and Theorem 5 in [27], could still prove to be useful for handling self-suspending tasks (that do not use suspension-based locks) if *efficient* schedulability tests or methods for constructing sets of single-segment deferrable tasks can be found. However, such tests or transformations have not yet been obtained and the development of a precise and efficient schedulability test for self-suspending tasks remains an open problem.

Acknowledgements. We thank James H. Anderson and Raj Rajkumar for their comments on early drafts of this paper.

References

1 Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

2 Aaron Block, Hennadiy Leontyev, Björn B. Brandenburg, and James H. Anderson. A flexible real-time locking protocol for multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56. IEEE Computer Society, 2007. doi:10.1109/RTCSA.2007.8.

3 Björn B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 141–152. IEEE Computer Society, 2013. doi:10.1109/RTAS.2013.6531087.

4 Björn B. Brandenburg. Blocking optimality in distributed real-time locking protocols. *Leibniz Transactions on Embedded Systems*, 1(2):01:1–01:22, 2014. doi:10.4230/LITES-v001-i002-a001.

5 Björn B. Brandenburg and James H. Anderson. A comparison of the M-PCP, D-PCP, and FMLP on LITMUS$^{\mathrm{RT}}$. In *Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS)*, volume 5401 of *Lecture Notes in Computer Science*, pages 105–124. Springer, 2008. doi:10.1007/978-3-540-92221-6_9.

6 Björn B. Brandenburg and James H. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization proto-

cols in LITMUS^{RT}. In *Proceedings of the 14th IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 185–194. IEEE Computer Society, 2008. doi:10.1109/RTCSA.2008.13.

7 Jian-Jia Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS)*, pages 327–338. IEEE Computer Society, 2016.

8 Jian-Jia Chen. A note on the exact schedulability analysis for segmented self-suspending systems. *The Computing Research Repository (CoRR)*, abs/1605.00124, 2016. URL: http://arxiv.org/abs/1605.00124.

9 Jian-Jia Chen and Cong Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160. IEEE Computer Society, 2014. doi:10.1109/RTSS.2014.31.

10 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, Ragunathan Rajkumar, and Dionisio de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Department of Computer Science, TU Dortmund, 2016. URL: http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2016-chen-techreport-854.pdf.

11 Hiroyuki Chishiro and Nobuyuki Yamasaki. Global semi-fixed-priority scheduling on multiprocessors. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 218–223. IEEE Computer Society, 2011. doi:10.1109/RTCSA.2011.32.

12 Wen-Hung Huang and Jian-Jia Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1078–1083. IEEE, 2016. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=7459469.

13 Junsung Kim, Björn Andersson, Dionisio de Niz, and Ragunathan Rajkumar. Segment-fixed priority scheduling for self-suspending real-time tasks. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, pages 246–257. IEEE Computer Society, 2013. doi:10.1109/RTSS.2013.32.

14 Karthik Lakshmanan. *Scheduling and Synchronization for Multi-core Real-time Systems*. PhD thesis, Carnegie Mellon University, 2011. URL: https://www.ece.cmu.edu/research/publications/2011/CMU-ECE-2011-040.pdf.

15 Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 469–478. IEEE Computer Society, 2009. doi:10.1109/RTSS.2009.51.

16 Karthik Lakshmanan and Ragunathan Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12. IEEE Computer Society, 2010. doi:10.1109/RTAS.2010.38.

17 John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the 8th IEEE Real-Time Systems Symposium (RTSS)*, pages 261–270. IEEE Computer Society, 1987.

18 John P. Lehoczky, Lui Sha, Jay K. Strosnider, and Hideyuki Tokuda. Fixed priority scheduling theory for hard real-time systems. In Andreé van Tilborg and Gary Koob, editors, *Fondations of Real-Time Computing: Scheduling and Resource Management*, chapter 1, pages 1–30. Kluwer Academic Publishers, 1991.

19 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

20 Cong Liu and James H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 425–436. IEEE Computer Society, 2009. doi:10.1109/RTSS.2009.10.

21 Cong Liu and James H. Anderson. Improving the schedulability of sporadic self-suspending soft real-time multiprocessor task systems. In *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 13–22. IEEE Computer Society, 2010. doi:10.1109/RTCSA.2010.14.

22 Cong Liu and James H. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 23–32. IEEE Computer Society, 2010. doi:10.1109/RTAS.2010.12.

23 Cong Liu and Jian-Jia Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 173–183. IEEE Computer Society, 2014. doi:10.1109/RTSS.2014.10.

24 Aloysius Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983. URL: https://dspace.mit.edu/handle/1721.1/15670.

25 Geoffrey Nelissen, José Fonseca, Gurulingesh Raravi, and Vincent Nélis. Timing analysis of fixed priority self-suspending sporadic tasks. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89. IEEE Computer Society, 2015. doi:10.1109/ECRTS.2015.15.

26 Ragunathan Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS)*, pages 116–123. IEEE Computer Society, 1990. doi:10.1109/ICDCS.1990.89257.

27 Ragunathan Rajkumar. Dealing with suspending periodic tasks. Technical report, IBM T. J. Watson Research Center, 1991.

28 Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

29 Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS)*, pages 259–269. IEEE Computer Society, 1988. doi:10.1109/REAL.1988.51121.

30 Frédéric Ridouard, Pascal Richard, and Francis Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*, pages 47–56. IEEE Computer Society, 2004. doi:10.1109/REAL.2004.35.

31 Jay K. Strosnider, John P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995. doi:10.1109/12.368008.

32 Jun Sun and Jane W.-S. Liu. Synchronization protocols in distributed real-time systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS)*, pages 38–45. IEEE Computer Society, 1996. doi:10.1109/ICDCS.1996.507899.

Utility-Based Scheduling of (m, k)-firm Real-Time Tasks – New Empirical Results

Florian Kluge*

Department of Computer Science, University of Augsburg, Germany
fkuau@gmx.net

Abstract

The concept of a firm real-time task includes the notion of a firm deadline that should not be missed by the jobs of this task. If a deadline miss occurs, the concerned job yields no value to the system. For some applications domains, this restrictive notion can be relaxed. For example, robust control systems can tolerate that single executions of a control loop miss their deadlines, and still yield an acceptable behaviour. Thus, systems can be designed under more optimistic assumptions, e.g. by allowing overloads. However, care must be taken that deadline misses do not accumulate. This restriction can be expressed by the model of (m, k)-firm real-time tasks that require that from any k consecutive jobs at least m are executed successfully. In this article, we extend our prior work on the MKU scheduling heuristic. MKU is based on history-cognisant utility functions as means for making decisions in overload situations. We present new theoretical results on MKU and other schedulers for (m, k)-firm real-time tasks. Based on extensive simulations, we assess the performance of these schedulers. The results allow us to identify task set characteristics that can be used as guidelines for choosing a scheduler for a concrete use case.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms
Keywords and phrases Real-time Scheduling, (m, k)-Firm Real-Time Tasks
Digital Object Identifier 10.4230/LITES-v004-i001-a002
Received 2016-01-25 **Accepted** 2016-12-28 **Published** 2016-12-31

1 Introduction

Certain types of real-time systems can tolerate that some jobs miss their deadlines or are not executed at all. This allows to dimension the system more optimistically. Sporadically arising overload conditions are resolved either by deferring or cancelling some jobs. Consider, for example, the decoding of a video stream. If single frames are displayed too late, the quality a viewer experiences degrades, but he still can draw some benefit. Similarly, control systems can also tolerate some job losses due to their robustness. To convey the notion of such systems with relaxed real-time constraints into real-time scheduling, Jensen et al. [19] and Locke [33] replaced the binary notion of deadlines with more expressive *time-utility function*s (TUFs) and proposed a scheduler based on *earliest deadline first* (EDF) [32]. A TUF describes the value or utility a system can draw from a job execution if it is finished by a certain time, thus increasing the flexibility of real-time systems.

A problem in TUF-based real-time scheduling is that each job is viewed independently. Therefore, no guarantees can be given about the distribution of deadline misses or job cancellations (both termed *losses* in the following) for single tasks. It may even happen that jobs of a specific task are never executed [24]. Considering the above examples, it is obviously necessary that losses of jobs do not accumulate and thus a simple *Quality-of-Service* (QoS) metric is not sufficient to describe the available tolerances. Special concepts have been developed in scheduling theory

* Florian Kluge is now with Elektronische Fahrwerkssysteme GmbH.

that allow to constrain the distribution of deadline misses, for example the skip-over model [27], (m, k)-firm real-time tasks [18], the dynamic window-constrained scheduler [44], or weakly-hard real-time tasks [4]. All these approaches consider not only single jobs, but also the execution history of the related tasks.

Our aim is to exploit the flexibility of TUF-based real-time scheduling while concurrently providing (m, k)-firm real-time guarantees. We use *history-cognisant utility functions* (HCUFs) [24] derived from TUFs to convey a task's state to a TUF-based real-time scheduler. A *history-cognisant utility function* (HCUF) represents the utility a task has accumulated with respect to the execution of past jobs. In our previous work [25] we have proposed a heuristic algorithm for *utility-based scheduling of* (m, k)*-firm real-time tasks* (MKU). The MKU algorithm is an extension of Jensen et al.'s [19, 33] EDF-based scheduler. Our results in [25] show the feasibility of MKU and that it can achieve good results when compared to other schedulers for (m, k)-firm real-time tasks.

In this article, we make the following contributions: First, we report new theoretical properties on preemptive scheduling of (m, k)-firm real-time tasks. A schedulability test for tasks using fixed (m, k)-patterns is presented. Additionally, we examine the phenomenon of breakdown anomalies, where increasing the utilisation of an infeasible task set can lead to feasibility. Also, new results on the schedulability of MKU are reported. Second, we present new results of extensive simulations that enable us to assess different schedulers for (m, k)-firm real-time tasks more clearly. We use arbitrary task sets to examine the overall performance of different schedulers, the feasibility of schedulability tests, and the initialisation of a task's execution history. In further simulations, we examine tasks sets where task periods and (m, k)-constraints are restricted to practically relevant ranges.

The remainder of this article is structured as follows: In the following Section 2, we define the basic concepts used throughout this paper. Related work on TUF-based scheduling and scheduling of (m, k)-firm real-time tasks is presented in Section 3. In Section 4, we present new properties of (m, k)-firm schedules which we use in our evaluations. In Section 5, we introduce our evaluation methodology. Evaluation results are shown and discussed in Section 6. We conclude this article in Section 7.

2 Fundamentals

2.1 Task Model

An (m, k)-firm real-time task is a tuple $\tau_i = (C_i, T_i, m_i, k_i)$ with *worst-case execution time* (WCET) C_i, period T_i and (m, k)-constraint (m_i, k_i). All numbers are integers. For simulation, we assume that a task's execution time is constant. In reality, the actual execution time of a task may be lower than its WCET. We account for this fact in our simulation through the use of abstract task sets and the generation of concrete task sets for different utilisations (see Section 2.2). Tasks are initially released at time $t = 0$, i.e. the task set is synchronous, and have implicit deadlines $D_i = T_i$. Thus, jobs $\tau_{i,j}$ are generated at times $r_{i,j} = jT_i, j = 0, 1, \ldots$ and must be finished until $d_{i,j} = (j + 1)T_i$ to avoid deadline misses. Each job is subject to a firm real-time requirement: If the job is not finished by its deadline, its result is useless and the job is cancelled. In this work, we consider the preemptive scheduling of jobs on a single processor. Thereby, we assume that jobs can be scheduled independently and that no resource constraints exist. If more than one job is eligible for dispatching at a certain time, e.g. due to identical priorities, then the scheduler chooses the job with the earliest activation time. If there are still multiple eligible jobs, an arbitrary one is chosen.

A task τ_i's (m, k)-constraint is defined by (m_i, k_i), meaning that in any k_i jobs released consecutively at least m_i must be finished before their deadline. An (m, k)-firm real-time task

incurs a *dynamic failure* if less than m out of k consecutive jobs meet their deadline. More formally, this can be expressed using the concept of a *k-sequence* of a task. Let $\sigma_i^j \in \{0,1\}$ denote the status of the j-th job of τ_i with $\sigma_i^j = 0$ representing a deadline miss or job cancellation, and $\sigma_i^j = 1$ standing for successful execution. Then, τ_i's state or *k-sequence* after execution of the j-th job is a string $\sigma_i = (\sigma_i^{j-k+1}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$ with $\sigma_i^l \in \{0,1\}^k$. New job states σ_i^j are shifted into σ_i from the right. The (m,k)-constraint requires that a task τ_i's k-sequence always contains at least m_i 1s. A task's *distance* from dynamic failure is the number of jobs that consecutively would have to miss their deadlines such that the task's (m,k)-constraint is no longer kept.

2.2 Abstract and Concrete Task Sets

*Abstract task set*s (ATSs) form the basis of the simulations presented in this article. An ATS $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is a set of abstract tasks $\alpha_i = (e_i, T_i, m_i, k_i)$ with periods T_i, (m,k)-constraints (m_i, k_i) and execution time weights e_i. A concrete task set $\tau(\alpha, U_\mathrm{T})$ is derived from an ATS α by calculating the tasks' execution times C_i such that the CTS approximates a target utilisation of U_T. The execution time weight e_i specifies, how much task τ_i contributes to the task set utilisation:

$$\frac{e_i}{\sum_{j=1}^{n} e_j} = \frac{U_i}{U_\mathrm{T}} \tag{1}$$

Thereby, $U_i = \frac{C_i}{T_i}$ is the utilisation of the task under consideration. Solving with eq. (1) for C_i yields:

$$C_i = \frac{U_\mathrm{T}}{\sum_{j=1}^{n} e_j} T_i e_i \tag{2}$$

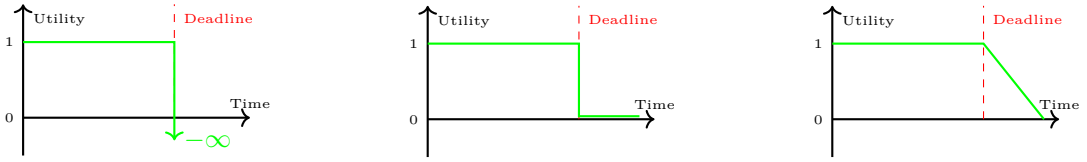
We consider only integral execution times in our simulations. How we obtain these will be clarified in Section 5.1.

3 Related Work

3.1 TUF-based Scheduling

The concept of time-utility functions was originally introduced by Jensen et al. [19] and Locke [33]. Instead of basing task scheduling solely on the binary notion of a deadline, the use of TUFs allows for a greater flexibility. A TUF describes the utility a system gains when a job is finished until a certain time. Some TUFs for well-known real-time constraints are shown in Figure 1. Hard real-time jobs (Figure 1a) must be finished by their deadline. Exceeding hard deadlines can result in catastrophic consequences which is expressed by the value $-\infty$ after the deadline. In contrast, firm real-time jobs (Figure 1b) do not yield any utility when exceeding their deadline. The entries in the k-sequence of a (m,k)-firm real-time task can be seen as the results of a firm real-time TUF. Soft real-time jobs may exceed their deadlines and still yield a decreasing utility to the system, which is shown in Figure 1c. TUFs are not restricted to these shapes. Thus, TUFs define a generic interface for the specification of task timing requirements. They allow to integrate tasks with different timing requirements in a system using only a single scheduler.

Jensen et al. [19] demonstrate the benefit of TUFs by extending EDF scheduling for overloaded task sets. If a high probability for a deadline miss is detected that would render the EDF schedule infeasible, jobs that contribute only with a low value-density (ratio of utility/value to execution time) to the system are selectively cancelled. Thus, schedulability of the system is ensured and



(a) Hard real-time TUF. **(b)** Firm real-time TUF. **(c)** Soft real-time TUF.

Figure 1 Exemplary TUFs.

accumulated utility is maximised. In literature, this approach is often referred to as *Locke's best-effort scheduling algorithm* (LBESA). Based on LBESA, Clark [11] has developed the *dependent activities scheduling algorithm* (DASA) for tasks with dependencies. Davis et al. [12] proposed an *adaptive threshold policy* for the admission of jobs, which has a lower overhead than LBESA. The notion of *dynamic value density* [1] reduces cancellations of jobs that have already started execution. Li and Ravindran [30] presented the MLBESA and MDASA algorithms that mimic the behaviour of LBESA and DASA, but come with lower complexities. TUF-based approaches have often been proposed to handle transient overloads in real-time systems [3, 26, 6, 35, 34, 13].

Many works on scheduling based on TUFs can also be found under the term *utility accrual* scheduling. The aim in utility accrual scheduling is to maximise the utility that is accrued through the execution of tasks. Insofar, the values and shapes of TUFs are a central criterion for scheduling. Chen and Muhlethaler [7] have shown that the problem of maximising value through arrangement of jobs/tasks is NP-hard. They also proposed an heuristic scheduling algorithm with a complexity of $O(n^3)$. The *utility accrual packet scheduling algorithm* by Wang and Ravindran [43] for packet scheduling in switched Ethernet comes with a lower complexity of $O(n^2)$, but is restricted to unimodal non-increasing TUFs. In contrast, the *resource-constrained utility accrual* algorithm by Wu et al. [45] can handle arbitrary TUFs and resource constraints at a complexity of $O(n^2 \log n)$. The *generic utility scheduling* algorithm by Li et al. [31] can also deal with mutual exclusion constraints, although with higher complexities of $O(n^3)$ for dispatching and $O(n^4 r)$ for scheduling. Tidwell et al. [42] model the scheduling problem as a Markov decision process that is solved offline and yields an optimal solution. The solution is used to generate a lookup table that is evaluated by an online scheduler in linear complexity. Also, works exist that investigate the use of TUF-based scheduling on multiprocessor systems [41, 10, 9, 39]. Here, especially the work of Rhu et al. [39] on the *global multiprocessor utility accrual scheduling algorithm for* (m, k)*-firm deadline-constrained multimedia streams* (gMUA-MK) algorithm is interesting, as they aim to schedule tasks with (m, k)-firm deadlines and TUFs on multiprocessors.

3.2 (m, k)-firm Real-Time Tasks

The concept of real-time tasks with (m, k)-firm deadlines was originally proposed by Hamdaoui and Ramanathan [18]. They present a scheme for *distance-based priority* (DBP) assignment of newly generated jobs for fixed-priority scheduling. Using this scheme, the priority of a job is set depending on the task's distance from dynamic failure. Jobs that belong to a task with a short distance are assigned higher priorities. The corresponding calculations are based on the task's k-sequence (see Section 2.1). Goossens [16] points out two properties of the DBP approach and devises an exact schedulability test. The first property concerns the initialisation of the k-sequences: Goossens shows that the string $\sigma_i = 1^k$ is not optimal under DBP and may yield to an infeasible schedule. In contrast, using an error state to initialise σ_i can result in a feasible schedule. The second property is the periodicity of feasible DBP schedules. Scheduling

decisions under DBP only depend on the (m,k)-constraints and k-sequences σ_i of the tasks, whose space is bounded. Let $P = \text{lcm}\{T_i \mid i = 1,\dots,n\}$ be the hyperperiod of the task set $\tau = \{\tau_i = (C_i, T_i, m_i, k_i) \mid i = 1,\dots,n\}$. As τ is a synchronous task set with implicit deadlines, at times $B = kP, k \in \mathbb{N}$ all jobs that were activated before t are finished, and each task releases a new job. Thus, the system is in the same state each $B = kP$, and only the k-sequences of the tasks may differ. For any task τ_i, $\sum_{j=m_i}^{k_i} \binom{k_i}{j}$ distinct k-sequences exist. The period of a feasible DBP schedule is bounded by

$$F = \prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P \tag{3}$$

as any combination of tasks and their k-sequences must be considered. Thus, if a task set τ is feasible in the interval $[0, F)$, i.e. no (m,k)-constraint is violated, then τ is always feasible. The exact schedulability test for τ consist of executing or simulating τ for this time interval and checking whether the (m,k)-constraints of all tasks are always kept. Once an (m,k)-constraint is violated, the test stops and returns that τ is not feasible. The test can be sped up by evaluating the *system state* $\sigma = (\sigma_1,\dots,\sigma_n)$ consisting of all tasks' k-sequences at each hyperperiod boundary $B = kP, k \in \mathbb{N}$. If a certain system state σ recurs, simulation can immediately be stopped, as the schedule will repeat itself and thus is feasible. We will term this optimised test in the following as *Goossens' schedulability test* (GST).

The seminal work of Hamdaoui and Ramanathan [18] has sparked a number of further works on the scheduling of (m,k)-firm real-time tasks. Ramanathan uses the concept of (m,k)-firm real-time tasks for the specific use case of control systems [38]. A deterministic classification into mandatory and optional jobs is proposed based on static (m,k)-patterns. Mandatory jobs are scheduled with their original, e.g. rate-monotonic [32] priority while optional jobs get the lowest possible priority. In the following, we will refer to this approach as *evenly distributed* (m,k)*-patterns* (MKP). A set of (m,k)-firm real-time tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ is considered feasible, if at least all mandatory jobs can be executed successfully. A schedulability test is based on the fact that the first instance $\tau_{i,0}$ of any task τ_i is always classified as mandatory. Ramanathan [38] provides a sufficient schedulability condition. However, this condition contains a timing non-deterministic term which makes it hard to evaluate [20]. Jia et al. [20] propose a schedulability test, which basically implements the response time analysis [2] for the first job of any task heeding the (m,k)-patterns. For task sets with harmonic periods, the test provides exact results, for all other task sets it is only sufficient.

Quan an Hu [37] note that the classification according to [38] introduces a high pessimism into the schedulability analysis, as at time $t = 0$ a mandatory job from any task in a task set gets ready. They relieve this critical instant by introducing spin or rotation values s_i that rotate the (m,k)-patterns of each task τ_i by s_i places. Quan and Hu propose a heuristic algorithm for finding good spin parameters, and also examine the use of a genetic algorithm for the determination of spin parameters. In this work, we will use the heuristic algorithm under the term *evenly distributed* (m,k)*-patterns with spin values* (MKP-S). As the original presentation of the algorithm in [37] is missing important information, we apply the corrections that we describe in [22]. Spin parameters are also considered by Semprebom et al. [40] for global time slot allocation in wireless real-time networks. Additionally, the authors propose an online schedulability/admission test.

Flavia et al. [14] extend the work of Ramanathan [38] on the use of (m,k)-firm real-time tasks for control of plants. They present a method to determine offline an optimal k_i parameter for a given controller and calculate controller parameters for all $m_i \in [1 \dots k_i]$. Depending on the actual plant state during runtime, optimal m_i are chosen and the controller parameters are set appropriately.

Cho et al. devise the *guaranteed dynamic priority assignment* (GDPA) scheme [8]. It is based on EDF scheduling, but additionally takes the tasks' distance from a failing state into account. Its aim is (1) to provide a bounded probability of violations of the (m, k)-firm constraints, and (2) to maximise the probability of kept deadlines. Under GDPA, ready jobs are not directly inserted into the EDF schedule, instead they are kept in ready list. Any time the schedule needs to be adjusted, each job's distance from dynamic failure is calculated. The EDF schedule is created by considering the jobs in increasing order of their distance from dynamic failure, i.e. critical jobs are preferred, and inserting them into the EDF schedule. If an insertion makes the schedule infeasible, the job is removed again from the schedule. Jobs in the ready list that are infeasible are cancelled. In the same paper, the *simplified guaranteed dynamic priority assignment* (GDPA-S) is proposed. It works similar to GDPA, but has a lower runtime complexity. GDPA-S keeps ready jobs in two lists, one in EDF order and another ordered in increasing distance from dynamic failure. Dispatching is performed either from the head of the EDF list, if the EDF schedule is feasible. Else, the most critical job (at the head of the second list) is dispatched. Again, infeasible jobs are cancelled immediately. Concerning non-preemptive scheduling of (m, k)-firm real-time tasks, the work on Matrix-DBP [36] provides necessary schedulability conditions. One that can also be applied to preemptive scheduling is based on the minimum load that is generated by a set of (m, k)-firm real-time tasks $\tau = \{\tau_1, \dots \tau_n\}$:

$$U_{\text{mk}} = \sum_{i=1}^{i=n} \frac{m_i}{k_i} \frac{C_i}{T_i} \tag{4}$$

The calculation of U_{mk} assumes that only m_i out of k_i jobs of any task τ_i are executed. If

$$U_{\text{mk}} > 1 \tag{5}$$

then τ is not feasible.

An approach similar to the concept of (m, k)-firm real-time tasks is proposed by Gettings et al. [15] for mixed-criticality systems with weakly-hard constraints. Their *adaptive mixed criticality – weakly hard* algorithm can skip up to s out of m consecutive jobs to reduce the load from low-critical tasks in a high-criticality mode, while still ensuring a guaranteed QoS for low-criticality tasks.

3.3 The MKU Algorithm

In a previous publication [25], we have presented the *utility-based scheduling of (m, k)-firm real-time tasks* (MKU) algorithm that is based on HCUFs [24, 23]. In the following, we give a brief outline of its functionality, please refer to [25] for more details. MKU is based on LBESA, the main difference is the utility function that is used in the decisions about job cancellations. We use a HCUF [24, 23], that maps the execution history of a task into a single scalar value and additionally provides a prediction about the tasks future utility. A task τ_i's current utility H_{m} after the completion or cancellation of its j-th job is the arithmetic mean of its current k-sequence (see Section 2.1). The value is additionally scaled by $\frac{k_i}{m_i}$ to enable an easy comparison between tasks with different (m, k)-constraints:

$$\hat{H}_{\text{m}}(\tau_i, j) = \frac{k_i}{m_i} \frac{1}{k_i} \sum_{l=0}^{k+1} \sigma_i^{j-l} = \frac{1}{m_i} \sum_{l=0}^{k_i-1} \sigma_i^{j-l} \tag{6}$$

Through the scaling, any task τ_i has the requirement that $\hat{H}_m(\tau_i, j) \geq 1$. For scheduling decisions, we use a task τ_i's potential utility $\hat{H}_{\text{P}}$ under the assumption that the currently active job $\tau_{i,j}$ is

cancelled and ignore the least recent job $\tau_{i,j-k_i+1}$:

$$\hat{H}_{\mathrm{p}}(\tau_i, j) = \frac{1}{m_i} \sum_{l=0}^{k_i-2} \sigma_i^{j-l} \tag{7}$$

If an overload situation occurs, the scheduler cancels jobs that have maximum $\hat{H}_{\mathrm{p}}$ values. To ensure the adherence of (m,k)-constraints, only jobs with $\hat{H}_{\mathrm{p}} > 1$ can be cancelled. If no job for cancellation can be found, the task set is infeasible.

As MKU is based on LBESA, it inherits the algorithm's complexity of $O(n^2)$ in overload situations. This is similar to the GDPA, GDPA-S and gMUA-MK approaches, but higher than for DBP and the pattern-based MKP/MKP-S schemes. However, unlike these schemes, MKU and the underlying LBESA are not restricted to (m,k)-firm real-time tasks. They allow the integration of tasks with other requirements in the same system, as long as these requirements can be expressed in terms of TUFs/HCUFs.

4 New Properties of (m,k)-firm Real-Time Tasks

In the following, we present some properties of (m,k)-firm real-time task sets that have not yet been reported in literature. First, we present exact schedulability conditions for approaches based on fixed (m,k)-patterns (Section 4.1) and for the MKU scheme (Section 4.2). Finally, we report our observations on scheduling anomalies in Section 4.3.

4.1 Feasibility of Approaches Based on Fixed (m,k)-Patterns

For (m,k)-firm real-time task sets that use fixed (m,k)-patterns defined in [38] (MKP), Jia et al. give a sufficient schedulability test [20]. As the test is only sufficient, it may reject some task sets that are actually feasible. Also, it cannot be applied to the MKP-S [37] approach, as the rotation of the patterns might move the critical instant of the task set. Nevertheless, an exact schedulability test can be derived for task sets that used fixed (m,k)-patterns. The derivation of this test is similar to the one for DBP scheduling [16] and uses the same preconditions: (1) The scheduling algorithm must be *deterministic*, and (2) it must be *memory-less*. In the context of fixed (m,k)-patterns, the second precondition means that the algorithm's decisions at any time depend only on static properties of the active tasks. In contrast to [16], the current k-sequence of a task has no influence on the schedule.

The exact schedulability test follows from the periodicity of schedules when using fixed (m,k)-patterns.

▶ **Theorem 1.** *Let a set of synchronous (m,k)-firm real-time tasks be scheduled by a fixed-priority scheduler. Priorities are derived from fixed (m,k)-patterns that classify jobs into mandatory and optional. Then the schedule is periodic with period*

$$P = \mathrm{lcm}\{k_i T_i \mid i = 1 \ldots n\}. \tag{8}$$

Proof. The priorities of jobs of a single task τ_i are derived from a fixed (m,k)-pattern of length k_i. Thus, each k_i jobs, i.e. after $k_i T_i$ cycles, the priority pattern recurs. For any two tasks τ_i, τ_j, the job and priority pattern generated by both recurs after $\mathrm{lcm}\{k_i T_i, k_j T_j\}$ cycles, as after this times both tasks' are in the same state as at the beginning (concerning their patterns). Via induction, this argument can be extended to n tasks $\tau_1, \ldots, \tau_n$. ◀

For a task set where the sufficient test by Jia et al. [20] does not indicate feasibility or where the test is not applicable (MKP-S), it suffices the simulate the schedule for at most P cycles (eq. (8)).

4.2 Schedulability under MKU

The schedulability test devised by Goossens [16] for task sets under DBP scheduling also applies to the MKU scheduler in terms of the upper bound for simulation. The test is solely based on the tasks' periods and (m, k)-constraints, and the fact that the DBP scheduler itself is memoryless. The MKU scheduler itself does not possess an internal state. Like DBP, it acts solely on the states of the tasks, namely their k-sequences to calculate a task's HCUF.

When applying GST for MKU scheduling, a further slight optimisation is possible. GST examines the system state $\sigma = (\sigma_1, \ldots, \sigma_n)$ after each hyperperiod. A repetition of σ without any task violating its (m, k)-constraint means that the schedule is cyclic and valid, as the schedule itself solely depends on σ. In its calculation of the possible HCUF $\hat{H}_\mathrm{p}$ (eq. (7)), the MKU scheduler only regards the most recent $k_i - 1$ entries of each σ_i. Insofar, it operates on a *reduced system state*:

▶ **Definition 2.** Let $\sigma^R = (\sigma_1^R, \ldots, \sigma_n^R)$ be the *reduced system state* of a set of (m, k)-firm real-time tasks. σ_i^R is obtained from a system state $\sigma_i = (\sigma_i^{j-k+1}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$ by ignoring the least recent entry, i.e. $\sigma_i^R = (\sigma_i^{j-k+2}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$.

The following theorem provides the basis for an optimisation of GST for the HCUF-based scheduler:

▶ **Theorem 3.** *If during the execution of GST with MKU a reduced system state σ^R recurs at a hyperperiod boundary, a cycle in the MKU schedule has been found.*

Proof. Let σ^1, σ^2 be two system states incurred in this order during execution of GST with MKU, such that for the derived system states $\sigma^{R,1} = \sigma^{R,2}$ (in the following simply σ^R. Further, let $L(\sigma) = (\sigma_1^{j-k+1}, \sigma_2^{j-k+1}, \ldots, \sigma_n^{j-k+1})$ be the vector of a system state σ's least recent entries that are ignored by σ^R. We can distinguish two cases:

1. If $L(\sigma^1) = L(\sigma^2)$, then also $\sigma^1 = \sigma^2$. The whole system state recurred, a cycle in the schedule has been found.
2. If $L(\sigma^1) \neq L(\sigma^2)$, then there exists at least one i such that $\sigma_i^{1,j-k_i+1} \neq \sigma_i^{2,j-k_i+1}$. However, the schedule S_1 produced by MKU between σ^1 and σ^2 solely depends on $\sigma^{R,1}$, as MKU regards only the σ_i^R for its cancellation decisions. Thus, MKU will produce the same schedule $S_2 = S_1$ after σ^2, as $\sigma^{R,1} = \sigma^{R,2}$. ◀

In case 1, a real cycle has been found, as is also detected by GST. Case 2 needs some closer inspection, as it can help to speed up the schedulability test:

▶ **Corollary 4.** *Let σ^1, σ^2 be two system states with $\sigma^{R,1} = \sigma^{R,2}$, $L(\sigma^1) \neq L(\sigma^2)$ (case 2 in the proof of Theorem 3), $\sigma^R := \sigma^{R,1} (= \sigma^{R,2})$, and σ^2 occurs after σ^1. Further, let σ^3 be the next system state with $\sigma^{R,3} = \sigma^R$. Then, $L(\sigma^3) = L(\sigma^2)$ and $\sigma^3 = \sigma^2$.*

Proof. Recall that the schedule S_1 produced by MKU after σ^1 only depends on $\sigma^{R,1} = \sigma^R$. Thus, MKU will produce the same schedule $S_2 = S_1$ after σ^2. As the decisions for S_1 and the result $\sigma^{R,2}$ are solely based on $\sigma^{R,1}$, S_2 (taking the same decisions) will produce the same result $\sigma^3 = \sigma^2$, and thus $L(\sigma^3) = L(\sigma^2)$. ◀

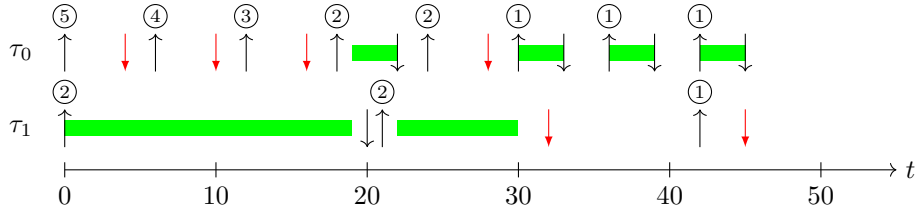
This means that the schedulability test for MKU **may** terminate at least one hyperperiod earlier compared to GST, depending on the length of the cycle.

4.3 Breakdown Anomalies

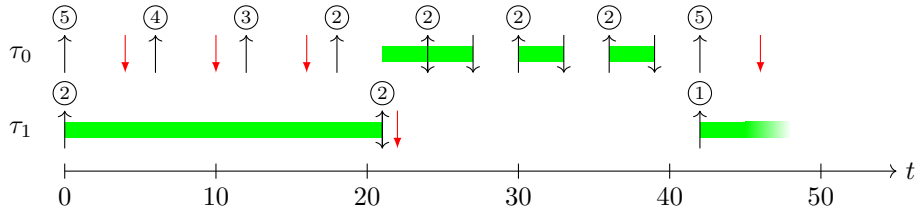
Consider an ATS (see Section 2.2), from which multiple CTSs are derived with increasing utilisations U. In hard real-time scheduling, it is possible to identify a *breakdown utilisation* [29] for an ATS,

Table 1 This ATS exhibits a breakdown anomaly at the target utilisations $U_{\mathrm{T}} = 1.45$ and $U_{\mathrm{T}} = 1.55$ when scheduled with DBP.

Task	T_i	e_i	(m,k)	$C_i^{1.45}$	$C_i^{1.55}$
τ_0	6	55	$(4,8)$	3	3
τ_1	21	95	$(1,2)$	19	21



(a) $U_{\mathrm{T}} = 1.45$, Task 1 violates its (1,2)-constraint at time 45.



(b) $U_{\mathrm{T}} = 1.55$, task set is feasible.

Figure 2 Example schedules with breakdown anomaly; for task parameters refer to Table 1; circled numbers (ⓧ) denote DBP of job; red arrows (↓) indicate job cancellations.

beyond which the derived CTSs are no longer feasible. Unfortunately, this method does not yield exact results for (m,k)-firm real-time tasks. Increasing the utilisation of a (m,k)-firm real-time task set farther beyond the breakdown utilisation can actually lead to the task set being feasible again. To make the point more clearly, consider an ATS α with a breakdown target utilisation U_{B}, and two constants $s_{\mathrm{I}} > 0, s_{\mathrm{F}} > 0, s_{\mathrm{I}} < s_{\mathrm{F}}$. This means that the derived CTS with target utilisation U_{B} is feasible, but the CTS with target utilisation $U_{\mathrm{B}} + s_{\mathrm{I}}$ is not. However, it may happen that the CTS with target utilisation $U_{\mathrm{B}} + s_{\mathrm{F}}$ is feasible again.

An exemplary ATS α that exhibits such a behaviour is shown in Table 1. We assume that both tasks' k-sequences are initialised with 1^{k_i}. The anomaly arises for target utilisations $U_{\mathrm{T}} = 1.45$ and $U_{\mathrm{T}} = 1.55$ (actual execution times are rounded to integer values). Figure 2a shows the DBP schedule for $\tau(\alpha, 1.45)$. Numbers in circles indicate the distance-based priority of each released job. Please refer to [38] for the calculations involved. Lower numbers indicate higher priorities. In time step 45, $\tau_{1,2}$ is cancelled, thus violating τ_1's $(1,2)$-constraint. Even if the scheduler cancelled the current instance of τ_0 instead, the schedule would still be infeasible, as then τ_0's $(4,8)$-constraint would be violated later. Increasing U_{T} for α leads to a feasible DBP schedule, as is shown in Figure 2b. In this case, the segment between $t = 0$ and $t = 42$ is repeated periodically.

The breakdown anomalies are the result of the special structure of the DBP scheme. Scheduling decisions made at a certain time do not only depend on static properties of the tasks. Instead, they are also influenced by the internal states of the tasks, which in turn depend on the past scheduling decisions. Thus, they can also occur in the MKU, GDPA and GDPA-S algorithms. The consequence of such a behaviour is that these schedulers are not sustainable [5] with regard to the

tasks' execution times. If a schedulability test shows that a certain task set is feasible, we cannot be sure that it stays feasible if execution constraints are relaxed by decreasing execution times.

Such anomalies cannot happen in task sets that are scheduled using fixed (m, k)-patterns, which we express in the following theorem:

▶ **Theorem 5.** *Let a set of (m, k)-firm real-time tasks τ^1 be derived from an ATS α for a given utilisation U_1. Further, assume that τ^1 is not feasible using fixed (m, k)-patterns. Then, any task set τ^2 derived from α for a utilisation $U_2 > U_1$ is also not feasible.*

Proof. Fixed (m, k)-patterns classify jobs into mandatory and optional jobs. A schedule S is considered feasible, if all mandatory jobs are executed successfully. Infeasibility of S means that at least one mandatory job $\tau_{i,j}$ misses its deadline at time $d_{i,j}$. Assume that $\tau_{i,j}$ is released at time $a_{i,j}$. Then, only mandatory jobs with priority higher than or equal to $\tau_{i,j}$'s priority are executed in the interval $[a_{i,j}, d_{i,j}]$. Increasing the utilisation of the task set means that the tasks' execution times are increased, but periods and thus activation times and deadlines remain unchanged. Thus, the processing time demanded by higher-priority jobs in $[a_{i,j}, d_{i,j}]$ can only further increase, and thus the deadline miss would occur also in the new task set. ◀

From this theorem follows that scheduling of tasks using fixed (m, k)-patterns is sustainable with regard to execution times.

5 Evaluation Methodology

We perform extensive simulations of randomly generated task sets to compare the scheduling approaches. The simulations are conducted using the `tms-sim` framework developed in our group [21], which is available as open source software. In this section, we present the methodology we apply in our evaluations.

5.1 Task Parameters

The parameters of the ATSs are generated using the `libc` pseudo-random number generator (`rand_r()`). For the task periods T_i, two approaches are implemented: During most simulations, the periods are chosen randomly from a given interval $\{T_{\min}, \ldots, T_{\max}\}$. Additionally, we have also implemented the period generator from Goossens and Macq [17]. This generator yields task periods that have many common divisors, and a limited hyperperiod compared to randomly chosen periods. Period generation is based on a matrix of multipliers, where each row contains powers of a prime number. Period generation randomly selects one entry from each row. The actual period then is the product of the chosen entries. For our simulations, we add a restriction that periods must be > 2 for any task.

Execution time weights are chosen from an interval $[1, e_{\max}]$ where $e_{\max}$ represents the granularity of the weights. Execution times C_i are calculated according to eq. (2). As we only consider integral execution times in this work, the actual execution time C_i' of a task τ_i is obtained from C_i through rounding. Additionally, we demand that no task has zero execution time:

$$C_i' = \begin{cases} [C_i], \text{ if } [C_i] > 0 \\ 1, \text{ else} \end{cases} \tag{9}$$

Thereby, the operation $[x]$ stands for regular rounding, i.e. returns the integer value that is nearest to x. Through the rounding, the task set's actual utilisation $U = \sum_{i=0}^{n} \frac{C_i}{T_i}$ can deviate from the target utilisation. Task set generation is configured such that generated ATSs that deviate more than a constant d_{U} from an initial target utilisation are automatically discarded.

Table 2 Task models, schedulers, and schedulability tests used in the experimental evaluation.

Model	Abbr.	Reference	Test
FPP Scheduler			
Distance-based priority	DBP	[18]	GST
Fixed (m,k)-patterns	MKP	[38]	[20], Sect. 4.1
MKP with pattern rotation	MKP-S	[37]	Sect. 4.1
EDF-based Schedulers			
Guaranteed Dynamic Priority Assignment	GDPA	[8]	GST
Simplified GDPA	GDPA-S	[8]	GST
Global Multiprocessor Utility Accrual scheduling for (m, k)-firm deadline-constraints	gMUA-MK	[39]	GST
Utility-based (m,k)-tasks	MKU	[25]	GST

The k_i parameters are chosen from an interval $\{k_{\min}, \ldots, k_{\max}\}$. For the m_i parameters, we have again implemented two approaches: Either, they can be chosen from $\{1, \ldots, k_i\}$. This approach can yield m_i values (compared to k_i) that can seem quite unrealistic. Therefore, we allow to limit the m_i to meaningful ranges. An additional parameter $r_{\mathrm{m}} \in [0,1]$ can be specified to lower-bound m_i. The actual m_i parameter then is chosen from $\{[r_{\mathrm{m}} k_i], \ldots, k_i\}$.

To the best of our knowledge, there is not yet an efficient way to find good intialisations for the k-sequences. Checking all possible initialisation values is not feasible, as $2^{\sum_{i=1}^{n} k_i}$ schedules would have to be examined. Therefore, we use 1^{k_i} as initial k-sequence, which might be as good or bad as any other (possibly random) choice.

5.2 Simulation

Simulations are performed using the exact schedulability tests. For MKP and MKP-S, we use the methods described by Jia et al. [20] (sufficient condition) and the one introduced in Section 4.1. For all other schedulers, GST [16] is applied. Simulation of a CTS is performed as deemed necessary by the schedulability tests. In the simulations, we search for the breakdown utilisations [29] of ATSs under different schedulers. This search works as follows: A single ATS is repeatedly used to generate CTSs. The first ATS is generated using a target utilisation $U_{\mathrm{T}} = U_{\mathrm{B}}$. If the CTS is found to be schedulable for a certain scheduler, U_{T} is increased by a utilisation step s_{U}. Using this updated U_{T}, a new CTS is derived from the ATS and another simulation is performed. This process is repeated until the CTS is no longer schedulable. The last U_{T} that yields a feasible CTS is called the *breakdown utilisation*. To account for breakdown anomalies (see Section 4.3), U_{T} is increased further and the derived CTSs are simulated, too. This process stops when the derived CTSs no longer fulfil the necessary schedulability condition $U_{\mathrm{mk}} \leq 1$ (see eq. (5)).

An overview of the task models and schedulers used for evaluation can be found in Table 2. For our simulations, we have adjusted the gMUA-MK approach to immediately cancel jobs that are removed from a schedule due an overload. Due to the assumption of constant execution times, they would be cancelled anyway. Like in the MKU approach, we use the TUF for firm real-time tasks (see Figure 1b).

Table 3 Parameters for task set generation and simulation.

Symbol	Description	Value
$\{T_{\min}, \ldots, T_{\max}\}$	Range for task periods (ignored when Goossens' and Macq's period generator [17] is used, sect. 6.2.2, 6.2.3)	$\{5, \ldots, 60\}$
$e_{\max}$	Granularity of execution time weights	100
$\{k_{\min}, \ldots, k_{\max}\}$	Range for the k_i parameter	$\{2, \ldots, 10\}$
$\{m_{\min}, \ldots, m_{\max}\}$	Range for m_i paramters	$\{2, \ldots, k_i\}$
r_{m}	If specified, restricts m_i to $\{[r_{\mathrm{m}} k_i], \ldots, k_i\}$ (only sect. 6.2.1, 6.2.3)	$\{0.1, 0.2, \ldots, 0.9\}$
$U_{\mathrm{T}} = U_{\mathrm{B}}$	Target/base utilisation of generated task sets	1.05
d_{U}	Maximum allowed deviation from U_{T}	0.05
s_{U}	Utilisation step for breakdown utilisation search	0.01 / 0.1

5.3 Parameters & Aims of the Evaluation

5.3.1 Parameters

An overview of the parameters used for task set generation and simulation can be found in Table 3. They are passed to `tms-sim` via the command line or a parameter file. The period and k parameter ranges are chosen such as to be comparable with other works, e.g. [37]. The d_{U} parameter is only used during generation of an ATS. If the CTS generated for the base utilisation $U_{\mathrm{B}} = U_{\mathrm{T}}$ is not inside the interval $U_{\mathrm{T}} \pm d_{\mathrm{U}}$, the ATS is discarded. The r_{m} parameters are only used when mentioned explicitly. All other experiments are based on the predefined $\{m_{\min}, \ldots, m_{\max}\}$ interval. For a fine-grained analysis of the schedulers' behaviour, we set $s_{\mathrm{U}} = 0.01$, as this enables a good identification of breakdown anomalies. In a second round of simulations where we examine the m_i parameter and task periods in more detail, we use $s_{\mathrm{U}} = 0.1$.

5.3.2 Aims

In our experimental evaluations, we aim to answer the following questions:

1. Our prior results [25] (subject to consolidation) indicate remarkably performance differences between the different schedulers, when using the ratio of task sets that are feasible as a performance metric. How do the different schedulers compare against each other, when an exact schedulability test is applied (Section 6.1.1)?
2. How pessimistic is Goossens' feasibility interval (eq. (3), [16])? Even for small task periods, m_i and k_i values, the interval can get quite large. How high are the savings in terms of simulated time introduced through GST? Also, we examine the practical relevance of the optimised schedulability test for MKU that can be derived from Theorem 3 and Corollary 4. The results are presented in Section 6.1.2.
3. How relevant are breakdown anomalies (Section 6.1.3)?
4. As Goossens [16] shows, the initialisation of the k-sequences of tasks can impact the feasibility of a task set. However, it is open how to find good initialisation values. We explore the possibility of cross-initialisation of k-sequences between different schedulers: If a task set with given initial k-sequence is feasible only under one of two schedulers, simulation of the successful scheduler necessarily runs into a cycle of k-sequences when applying GST. Is it possible to use one of the recurring k-sequences as initial value for execution with the hitherto failing scheduler (Section 6.1.4)?

5. Due to cancellations, processing time already spent by the cancelled jobs is lost. How much performance is lost by the different schedulers (Section 6.1.5)?
6. In general, our evaluations are based on arbitrary task sets with random parameters which may not always have practical counterparts. If we restrict task periods and/or m parameters to realistic ranges/values, does this have any influence on the above questions? Both aspects are examined in Section 6.2.

6 Results

In [25], we have presented initial results on the performance of the some of the schedulers listed in Table 2. These results are based on the simulation of random task sets for a fixed number of time steps. If during this simulation time no violation of an (m, k)-constraint is detected, the task set is classified as feasible. This approach can yield false positive results, as an infeasibility may also happen just after the given number of time steps. Nevertheless, these results gave a first impression of the performance of the schedulers: Best results were achieved with the DBP and MKU approaches, followed by MKP and MKP-S. Least performance was exhibited by GDPA. Due to a bug in the implementation of the simulator, approaches based on FPP scheduling (DBP, MKP, MKP-S) exhibited a lower performance in [25] than they actually have. Also, MKU showed better performance than DBP for moderate overloads. We will consolidate these results in the following by using the exact schedulability tests as appropriate for the different schedulers. The results presented in this section are based on two groups of simulations. In the first group (Section 6.1), arbitrary task sets are examined in order to answer the first five questions laid down in Section 5.3.2. The second group (Section 6.2) deals with the use of realistic task parameters (last question in Section 5.3.2). These are only examined from the performance point of view. A discussion of all results follows in Section 6.3.

6.1 Arbitrary Task Sets and Exact Schedulability Test

As already stated above, the simulation of a task set for a fixed number of time steps can yield false positive results. For a closer examination of the different approaches, we therefore apply the exact schedulability tests (see Table 2). We combine this with a search for the breakdown utilisation of an ATS (see Section 5.2). ATSs are executed beyond the breakdown point to account for breakdown anomalies (Section 4.3). CTSs derived from these ATSs are simulated as deemed necessary by the schedulability tests. The results presented in the following are base on the simulation of 500 ATSs that are executed with all schedulers in Table 2. Beyond performance ratings of the scheduling approaches, we also investigate the performance of GST, breakdown anomalies, and cross-initialisation of k-sequences between schedulers.

6.1.1 Scheduler Performance

The overall performance of all schedulers is shown in Figure 3. Due to breakdown anomalies, it may be possible that some ATSs are actually feasible again for higher utilisation, which is discussed later in Section 6.1.3. If we compare these numbers with the results of the rough estimation presented in [25], we note that the actual success ratio of all schedulers is lower, which is just to be expected due to false positives in the original results. Nevertheless, the ratios between the different schedulers remain nearly unchanged. The DBP and MKU approaches still exhibit outstanding performance. So, the rough estimation at least allows for qualitative comparison of the schedulers. Concerning the bug in the implementation of the FPP scheduler, the results now show that for

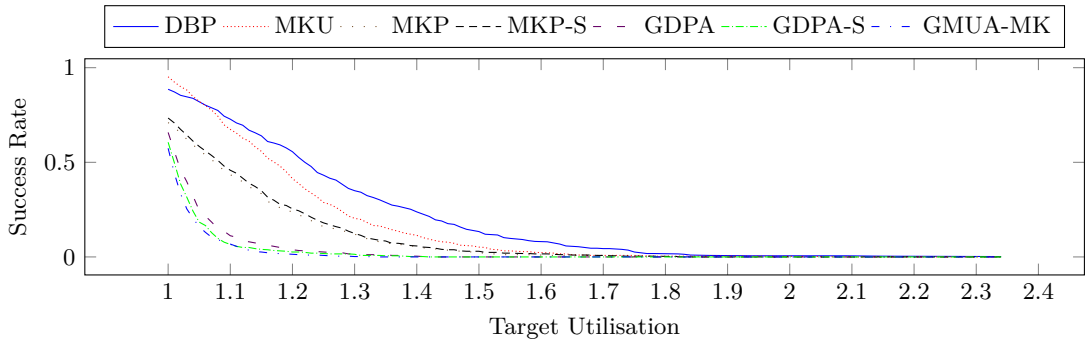


Figure 3 Ratio of ATSs that are schedulable up to a certain target utilisation U_{T}.

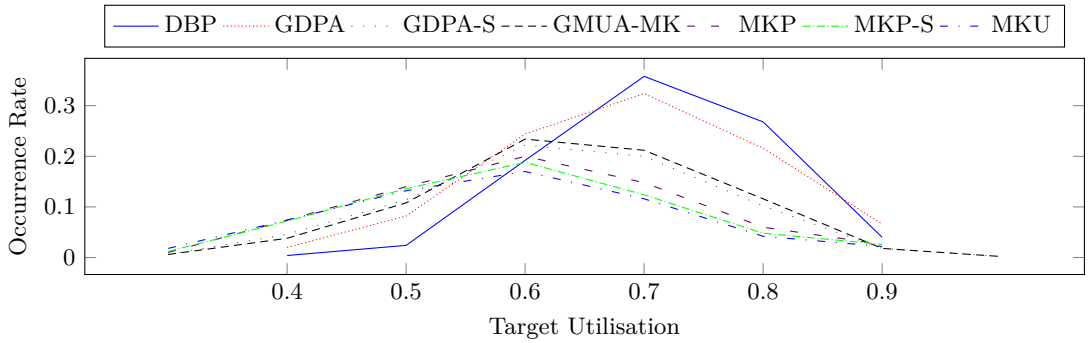


Figure 4 Breakdown (m, k)-utilisations U_{mk} (classified by rounding to nearest tenth).

most target utilisations DBP yields a better performance than MKU. Also, the MKP and MKP-S approaches exhibit a higher performance than estimated in [25].

Beyond providing a necessary schedulability condition, the (m, k)-utilisation U_{mk} does not help further to estimate the feasibility of a task set. Figure 4 shows the occurrence rate of the (m, k)-utilisations at the breakdown point of a task set, classified by rounding to the nearest tenth. The class $U_{\mathrm{mk}} = 0$ stands for ATSs that are not feasible at all. Obviously, a high (m, k)-utilisation does not per se prohibit feasibility, although this is achieved by only few ATSs. Most ATSs have a breakdown (m, k)-utilisation in the interval $[0.55, 0.85)$.

6.1.2 Performance of Schedulability Tests

The exact schedulability test for the approaches based on fixed (m, k)-patterns proposed in Section 4.1 yields a large feasibility interval. However, simulations must only be performed if the sufficient test [20] fails. In contrast, the feasibility interval for DBP [16] tends to exceed the MKP feasibility interval by far. In the following, we concentrate on the gains that are obtained through GST.

The DBP feasibility interval defines a very high bound for the number of hyperperiods that must be simulated successfully until feasibility of a task set can safely be assumed. Our experiments show that this bound is quite pessimistic and the optimisation incorporated in GST yields great value for the schedulability test. In all schedulers using GST, infeasibility is detected for most task sets ($> 99\,\%$) during the first hyperperiod, only few take longer. In the experiment at hand, the longest simulation to detect infeasibility takes three hyperperiods; in other simulations we observed durations of up to six hyperperiods. Feasibility is mostly found after the second hyperperiod,

Table 4 Number of ATSs that exhibit at least one breakdown anomaly.

	DBP	GDPA	GDPA-S	gMUA-MK	MKU
Absolute	20	11	9	11	40
Relative (%)	4.0	2.2	1.8	2.2	8.0

again with only few CTSs needing more time (up to 18 hyperperiods can be observed). For feasible CTSs, the first hyperperiod can be seen as a warm-up phase: At the start, the k-sequences have an arbitrary initialisation, in our case 1^k. These k-sequences are very unlikely to recur, as at least some jobs necessarily must be cancelled due to the overload. So, during the warm-up phase a good initialisation for the k-sequences is found, which leads into a recurring system state.

The gain from the optimised schedulability test for MKU that follows from Theorem 3 and Corollary 4 is only marginal. From 10270 CTSs in the experiment that are feasible under MKU, only 25 (0.2 %) would finish earlier.

6.1.3 Breakdown Anomalies

Extending the simulations described previously beyond the ATSs' breakdown points yields the following results. As already proven in Theorem 5, the schedulers based on fixed (m, k)-patterns (MKP, MKP-S) do not exhibit any anomalies. For the other schedulers, Table 4 gives the numbers of ATSs that exhibit at least one breakdown anomaly. With 8 % of the ATSs, MKU exhibits the largest number of anomalies. Thus, like all unsustainable algorithms, it should be treated with care.

6.1.4 Cross-Initialisation of k-Sequences

As noted by Goossens [16], the choice of the initial k-sequence can have significant impact on the feasibility of a task set. So far, no efficient algorithm is available that can derive a meaningful initialisation. We note that disjoint sets of CTS exist in our simulation results that are feasible only under one of any two approaches, but not under both. Feasibility in this case means that, after an initial warm-up phase which started with each task's k-sequence σ_i being initialised to 1^{k_i}, a system state $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ (see Section 3.2) periodically recurs. For a CTS that is feasible under two scheduling approaches, the corresponding system states may be different.

Our idea is to use a periodically recurring system state of a CTS that is only feasible under one of two schedulers for initialisation of the same CTS under the other scheduler. This approach might especially be interesting to improve the performance of approaches like GDPA. More formally, we assume a task set τ with $\sigma_i = 1^{k_i}, i = 1, 2, \ldots, n$ that is feasible under a scheduler $SC_\mathrm{F} \in \mathit{Schedulers} = \{\mathrm{DBP}, \mathrm{MKU}, \mathrm{GDPA}, \mathrm{GDPA\text{-}S}, \mathrm{gMUA\text{-}MK}\}$, but not under another scheduler $SC_\mathrm{I} \in \mathit{Schedulers}$. Thus, the execution of τ using SC_F finally runs into a cycle where a system state σ^F with $\sigma_i^\mathrm{F} \neq 1^{k_i}$ recurs periodically. This follows from the fact that the task set is overloaded and thus some jobs must be cancelled. Now we derive a task set τ' from τ by initialising each task τ_i''s k-sequence with the corresponding data from σ^F. τ' then is simulated under SC_I using GST to check whether the new initial k-sequence leads to a valid schedule. This approach is not applicable to the MKP and MKP-S approaches, as these disregard tasks' k-sequences.

In our simulations, we can identify significant numbers of candidate task sets only in the DBP and MKU approaches. Concerning the transfer of their final system state σ^F to other schedulers, only negligible successes can be achieved. For example, 263 distinct task sets are feasible in our simulations under DBP, but not under GDPA. Using their final system state for execution under

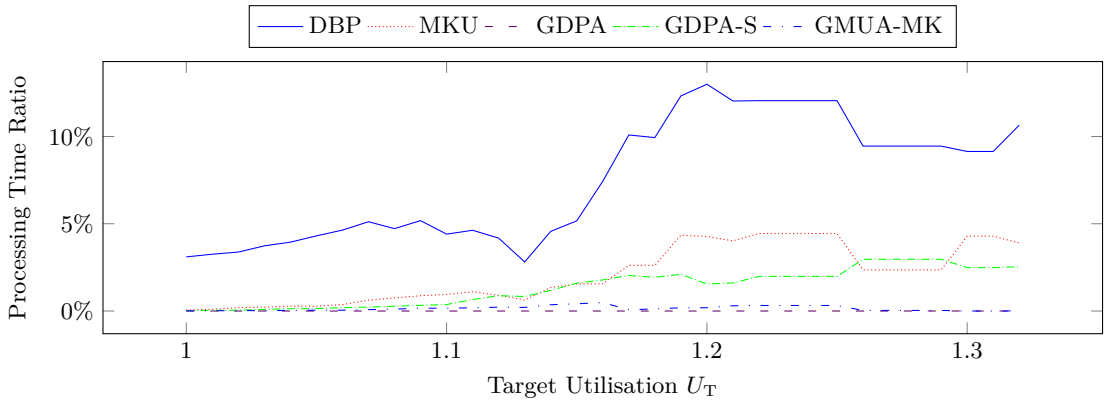


Figure 5 Mean lost processing time through EC (only feasible task sets; scaled by number of hyperperiods and hyperperiod length; only task sets where all schedulers successful).

GDPA leads to feasibility for only one task set. For a cross-initialisation from DBP to gMUA-MK, only 2 out of 274 candidates gain feasibility. Other transfers yield similar results. Thus, the cross-initialisation approach does not promise to lead to significant improvements concerning feasibility.

6.1.5 Cancellation of Running Jobs

Cancelling a job that has already started execution leads to the already consumed processing time being lost. Figure 5 shows the mean ratio of processing time that is lost due to cancellation of executing jobs. The schedulers based on fixed (m, k)-patterns (MKP, MKP-S) are omitted for two reasons: (1) If the sufficient schedulability test is successful, no simulation is performed, so no numbers are available for some task sets. (2) Only optional jobs (having lowest possible priority) are allowed to be cancelled; processing time that would be lost could be reclaimed by (possibly non-real-time) tasks that are running above the lowest possible priority, but still below the priorities of the (m, k)-firm real-time tasks.

Only CTSs that are feasible under all schedulers are included in the figures. The numbers are calculated in the following manner: For each CTS, the number of lost time steps is scaled by the task set's hyperperiod and the number of hyperperiods it is executed. From these numbers, the average is calculated for each target utilisation.

Figure 5 shows that approaches that have a rather low performance (in terms of their ability to find feasible schedules), tend to loose only a minor portion of processing time due to cancellation of already executing jobs. Most interestingly, in the GDPA no processing time is lost at all. This can be explained through the special technique in which GDPA calculates its schedule: Jobs with a high distance from dynamic failure are considered later for insertion into the EDF schedule than those that are near to a dynamic failure. If the insertion makes the EDF schedule infeasible, the job is removed again and deferred (but not yet cancelled!). Jobs with high distance to dynamic failure tend to be considered rather late for the schedule, and thus have a higher probability to lead to infeasibility, as the schedule might already be rather "full" through more critical jobs. Thus they are deferred without being executed, until they are cancelled due to missing their deadline.

The losses in the gMUA-MK approach stays well below 3 %. The costs incurred by GDPA-S are similar to those of MKU. Compared to the other approaches, DBP loses the highest amount of processing time.

The results allow us also to deduct that the higher flexibility of the DBP and MKU approaches (in terms of finding feasible schedules) is bought at the cost of a higher amount of lost processing

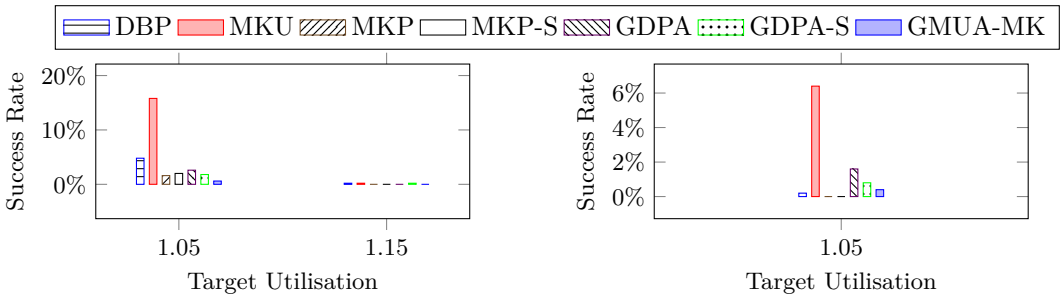


Figure 6 Performance with restricted m_i, $r_\mathrm{m} = 0.8$.

Figure 7 Performance with restricted m_i, $r_\mathrm{m} = 0.9$.

time (here up to 13 %). Thereby, lower costs are incurred by MKU. This is due to the fact that MKU cancels jobs in a more anticipatory manner as soon as an overload pends somewhere in the schedule. Jobs in DBP are only cancelled when they can no longer meet their deadline.

6.2 Realistic Periods and m Parameters

The results presented so far are based on task sets with quite arbitrary parameters, concerning especially the task periods and m parameters. In reality, one would not find such a great variability: In real applications, task periods within a task set can be tuned to be harmonic or at least have many common divisors, and they span several orders of magnitude (see e.g. [28]). Also, it seems unrealistic to have tasks with a low ratio $\frac{m}{k}$ which would mean that most jobs could be skipped. In the following, we examine the behaviour of DBP and MKU under more realistic conditions by restricting m parameters, task periods, and both. For all results, the utilisation step is set to $s_\mathrm{U} = 0.1$. Breakdown anomalies are ignored, i.e. an ATS is simulated only until the first infeasible CTS (these may differ for different schedulers!).

6.2.1 Restricted m Parameters

If generation of the m parameter of a task is restricted to an interval $[r_\mathrm{m} k_i, k_i]$, we get some interesting results. In the following evaluations, we examine values $r_\mathrm{m} \in \{0.1, 0.2, \ldots, 0.9\}$. Although, in our view, reasonable r_m values would rather be in the upper part of this set, we also need to look at low values, as we will see soon.

For each value of r_m, 500 ATSs are generated and simulated again with all schedulers. Like before, we use the breakdown utilisation of the ATSs and the number of feasible CTSs for each target utilisation to assess the performance of the schedulers. Breakdown anomalies occur in these simulations only rarely ($\leq 2\,\%$ of the ATSs per r_m value), and are therefore ignored.

We show the success rates of the schedulers for $r_\mathrm{m} = 0.8$ and $r_\mathrm{m} = 0.9$ in Figures 6 and 7. Further diagrams for the other r_m values can be found in appendix B. For most schedulers, the performance ratio between any two stays similar to that found in the above simulations. As expected, the overall performance decreases with increasing r_m. This can most clearly be observed by a decrease of the maximum target utilisation for which feasible CTSs exist. Also, the number of feasible CTSs at $U_\mathrm{T} = 1.05$ decreases rapidly with increasing r_m.

However, the DBP and MKU schedulers exhibit a more interesting behaviour, when examined in this detail compared to the simulations in 6.1. For very moderate overloads ($U_\mathrm{T} = 1.05$), MKU achieves in most simulations, where $r_\mathrm{m} \geq 0.3$, a better average performance than DBP. It seems that in these situations the advantages of EDF, which MKU is based on, over fixed-priority

$$\begin{pmatrix} 1 & 1 & 2 & 2 & 4 & 4 & 8 & 8 & 16 & 32 \\ 1 & 1 & 3 & 3 & 3 & 3 & 9 & 9 & 9 & 9 \\ 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 25 & 25 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 49 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 11 & 11 & 11 \end{pmatrix}$$

Figure 8 Matrix used for generation of realistic task periods.

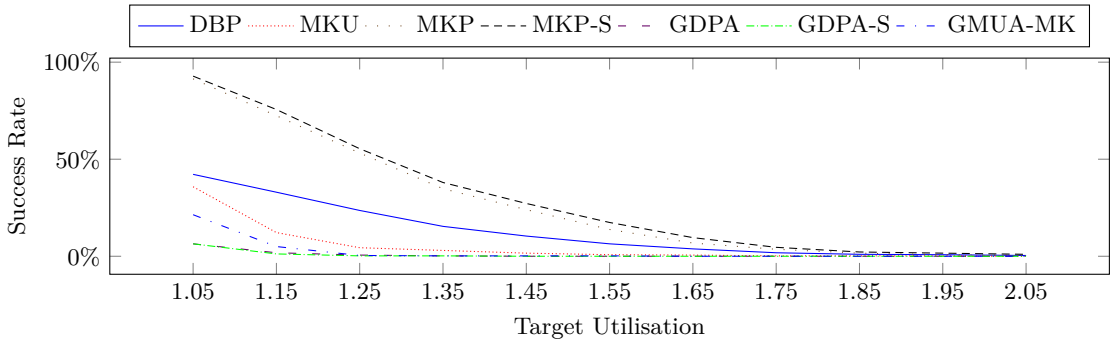


Figure 9 Ratio of task sets with realistic periods that are feasible up to a certain target utilisation U_{T}.

scheduling can still surface despite the overload. Applying the cross-initialisation technique (see Section 6.1.4) to transfer successful k-sequences from MKU to the more runtime-efficient DBP does not yield any successes for $r_{\mathrm{m}} \geq 0.6$, which in our view defines the most relevant range of m parameters.

6.2.2 Realistic Periods

In real applications, periods are often harmonic or have at least many common divisors. Also, they usually span several orders of magnitude. To imitate such circumstances, we use the period generator proposed by Goossens and Macq [17], which is aimed to generate task sets with limited hyperperiods (see Section 5.1). For our experiments, we use the matrix shown in Figure 8. With this matrix, we get periods in a range from 3 to 3,880,800. The maximum hyperperiod is also 3,880,800. All other task parameters are chosen as shown in Table 3.

Simulations are again performed for 500 ATSs. Figure 9 shows the ratio of ATSs that are feasible for a given target utilisation under the DBP and MKU approaches. Values for $U_{\mathrm{T}} > 2.05$ are omitted, as they are very small.

The larger range of periods seem to have a rather detrimental effect on most schedulers. Compared to the periods used in Section 6.1, their performance is more than halved. However, some exceptions exist: For a moderate overload at $U_{\mathrm{T}} = 1.05$, gMUA-MK actually improves, but deteriorates fast for larger U_{T}. The MKP and MKP-S schedulers actually achieve much better results. These can be attributed to the larger range of periods in the single task sets. As both approaches also use *rate monotonic* (RM) priorities [32] for their mandatory jobs, the critical instance at time $t = 0$ is greatly relieved: In the previous evaluations (Section 6.1), all jobs released at this time have similar deadlines and thus compete for processing time in the same interval. In contrast, with the extended period ranges, low-priority tasks (having long deadlines) can profit from multiple activations of tasks with shorter period within their period, as they can easily supersede the optional instances of the short-period tasks. Parts of the improvement may also stem from the fact that some task sets in this simulation have harmonic periods. For such

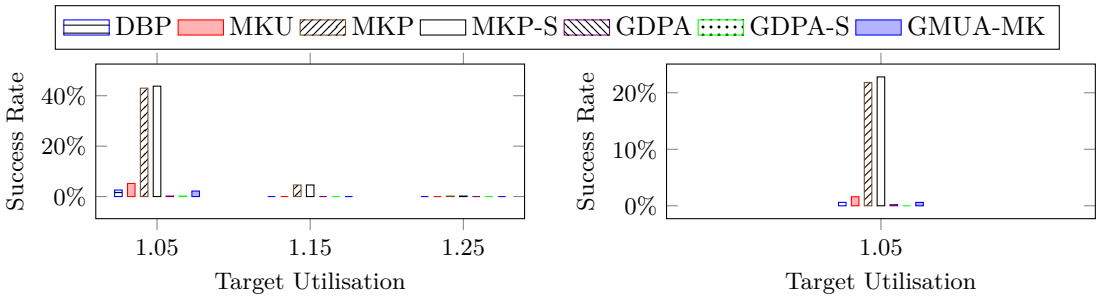


Figure 10 Performance with restricted m_i ($r_\mathrm{m} = 0.8$) and realistic periods.

Figure 11 Performance with restricted m_i ($r_\mathrm{m} = 0.9$.) and realistic periods

task sets, it is known that the utilisation bound for schedulability under fixed priorities increases to $U \leq 1.0$, compared to the Liu/Layland bound of $U \approx 0.69$. Concerning the deterioration of DBP, we note that DBP in many cases makes decisions that are sub-optimal for such task sets. Depending on (m, k)-constraints, it happens that DBP prefers a task with higher period over one with lower period due to the priority assignment being solely based on distance from dynamic failure. In such task sets, this often leads to multiple consecutive low-period jobs (with actually high RM priority) being not executed at all and thus a violation of (m, k)-constraints.

6.2.3 Combination

Finally, we examine the combination of restricting tasks' m parameters and periods. Task parameters are generated as in Section 6.2.1 except for the periods, for which we employ Goossens' and Macq's approach [17] already used in Section 6.2.2. Again, 500 ATSs are generated and simulated. Performance numbers are again based on the breakdown utilisations, ignoring the rarely occurring breakdown anomalies.

Exemplarily, we show the success rates of the schedulers for $r_\mathrm{m} = 0.8$ and $r_\mathrm{m} = 0.9$ in Figures 10 and 11. Further diagrams for the other r_m values can be found in appendix B. They can be interpreted as a combination of the results of the previous two experiments. The performance of all approaches is clearly dominated by the larger variance of periods. The MKP/MKP-S approaches still achieve outstanding performance due to the higher variation of task periods within the task sets. The other approaches suffer from both the regular periods and the high r_m parameter.

6.3 Discussion

Our results make several points in regard to which scheduler should be used for which kind of set of (m, k)-firm real-time tasks. First, if task periods span several orders of magnitude, as is often the case for industrial applications, the schedulers based on fixed (m, k)-patterns can achieve better performance (see Section 6.2.2). They have the additional advantage that a simple schedulability test [20] is available. If task periods can be tuned to be harmonic, the test yields exact results. Second, for task periods that are in the same order of magnitude, better results are achieved using one of the DBP or MKU schedulers (see Section 6.1). Depending on how strongly the task set is constrained by (m, k)-parameters, either one of the two tends to yield better results. If constraints are very harsh, i.e. if the m_i are very near to the k_i, then the performance tends to be higher under MKU, and vice versa for DBP. However, it may still happen that, e.g. a strongly constrained task set is feasible under DBP, but not under MKU. So the final choice of a scheduler must be based on a accurate examination of the task set considering all schedulers available.

7 Conclusions

In this article, we have extended our prior work on HCUF-based scheduling of (m, k)-firm real-time tasks and examined several schedulers for (m, k)-firm real-time tasks. For existing schedulers for (m, k)-firm real-time tasks, we pointed out some new properties, namely an exact schedulability test for scheduling based on fixed (m, k)-patterns and the existence of breakdown anomalies in approaches like DBP. Concerning our HCUF-based heuristic MKU, we presented new formal results on the schedulability.

In an experimental evaluation, we examined the schedulers under several points of view. Therefore, extensive simulations of randomly generated task sets were performed using the different schedulers and different generation approaches. The simulations are based on the search for breakdown utilisation [29] of abstract task sets. Our results show that the HCUF-based heuristic MKU can achieve a similar performance as DBP [18], which has the best performance among all schedulers regarded if task periods within a task set are roughly in the same order of magnitude and (m, k)-constraints are very heterogeneous. Both approaches were able to find feasible schedules for up to 80 % of the generated task sets. They also show the advantage of the optimisation that GST [16] introduces for testing the exact schedulability condition for (m, k)-firm real-time task sets under DBP, as GST can reduce the simulation time significantly. The results show further that no clear relation exists between a task sets' (m, k)-utilisation and its feasibility. The occurrence of breakdown anomalies in our results indicate that care must be taken when using one of the DBP, MKU, GDPA, GDPA-S or gMUA-MK schedulers for an actual system: if the actual execution time of tasks is smaller than assumed during schedulability analysis, the task set may become infeasible under these schedulers. Depending on the used scheduler, about 2-8 % of the task sets were affected by this problem in our simulations. We also tackled the problem of finding good initialisations of tasks' k-sequences, as these can impact feasibility [16]. Using results produced by a feasible schedule as initialisation for a task set under another scheduler, where it is infeasible so far, could yield only minor improvements. The examination of processing time lost due to job cancellations gives some surprising results: Under this metric, the GDPA/GDPA-S [8] and gMUA-MK [39] schedulers achieved best performance (less than 2 % loss), while DBP lost up to 13 % of the processing time. The MKU approach lies somewhere between these numbers.

In further simulations, we restricted task set generation to realistic parameters. A lower bound for the m parameter prohibited the generation of tasks whose jobs are scarcely executed. As could be expected, having the m_i parameters of tasks near their k_i parameters resulted in a decrease of all schedulers' performances. However, we also observed that in such a scenario with very strong (m, k)-constraints, MKU can actually achieve better results than DBP. By using the period generator proposed by Goossens and Macq [17], periods spanning multiple orders of magnitude inside a task set were generated. In such task sets, the performance of MKU and DBP degraded significantly due to sub-optimal decisions. Concurrently, the schedulers based on fixed (m, k)-patterns (MKP [38] and MKP-S [37]) could achieve much higher performance (feasibility for up to ≈ 93 % of the generated CTSs).

We draw the following conclusions from our results: When the periods in a task set span several orders of magnitude, as is e.g. the case for automotive systems [28], then an approach using fixed (m, k)-patterns should be preferred. When task periods are roughly the same order of magnitude, then DBP or MKU can yield a better performance, but care must be taken for schedulability anomalies, as both algorithms are not sustainable.

In this article, we restrict ourselves to mapping (m, k)-firm constraints with HCUF, but the scheduling approach is not limited to these (m, k)-HCUFs. We expect that also other HCUFs can be used in the future, e.g. to communicate applications' current state and requirements such that the scheduler can adjust its decisions.

References

1 Saud Ahmed Aldarmi and Alan Burns. Dynamic value-density for scheduling real-time systems. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 270–277. IEEE Computer Society, 1999. doi:10.1109/EMRTS.1999.777474.

2 Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

3 Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis E. Shasha. On-line scheduling in the presence of overload. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 100–110. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185354.

4 Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001. doi:10.1109/12.919277.

5 Alan Burns and Sanjoy K. Baruah. Sustainability in real-time scheduling. *JCSE*, 2(1):74–97, 2008. URL: http://jcse.kiise.org/PublishedPaper/year_abstract.asp?idx=15.

6 Giorgio C. Buttazzo, Marco Spuri, and Fabrizio Sensini. Value vs. deadline scheduling in overload conditions. In *16th IEEE Real-Time Systems Symposium, Palazzo dei Congressi, Via Matteotti, 1, Pisa, Italy, December 4-7, 1995, Proceedings*, pages 90–99. IEEE Computer Society, 1995. doi:10.1109/REAL.1995.495199.

7 Ken Chen and Paul Mühlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10(3):293–312, 1996. doi:10.1007/BF00383389.

8 Hyeonjoong Cho, Yongwha Chung, and Daihee Park. Guaranteed dynamic priority assignment scheme for streams with (m, k)-firm deadlines. *ETRI Journal*, 32(3):500–502, June 2010. doi:10.4218/etrij.10.0109.0544.

9 Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. Utility accrual real-time scheduling for multiprocessor embedded systems. *J. Parallel Distrib. Comput.*, 70(2):101–110, 2010. doi:10.1016/j.jpdc.2009.10.003.

10 Hyeonjoong Cho, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. On multiprocessor utility accrual real-time scheduling with statistical timing assurances. In Edwin Hsing-Mean Sha, Sung-Kook Han, Cheng-Zhong Xu, Moon-hae Kim, Laurence Tianruo Yang, and Bin Xiao, editors, *Embedded and Ubiquitous Computing, International Conference, EUC 2006, Seoul, Korea, August 1-4, 2006, Proceedings*, volume 4096 of *Lecture Notes in Computer Science*, pages 274–286. Springer, 2006. doi:10.1007/11802167_29.

11 Raymond Keith Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, August 1990.

12 Robert I. Davis, Sasikumar Punnekkat, Neil C. Audsley, and Alan Burns. Flexible scheduling for adaptable real-time systems. In *1st IEEE Real-Time Technology and Applications Symposium, Chicago, Illinois, USA, May 15-17, 1995*, pages 230–239. IEEE Computer Society, 1995. doi:10.1109/RTTAS.1995.516220.

13 Wanfu Ding and Ruifeng Guo. Design and evaluation of sectional real-time scheduling algorithms based on system load. In *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008, Zhang Jia Jie, Hunan, China, November 18-21, 2008*, pages 14–18. IEEE Computer Society, 2008. doi:10.1109/ICYCS.2008.208.

14 Felicioni Flavia, Jia Ning, Françoise Simonot-Lion, and Yeqiong Song. Optimal on-line (m, k)-firm constraint assignment for real-time control tasks based on plant state information. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*, pages 908–915. IEEE, 2008. doi:10.1109/ETFA.2008.4638504.

15 Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In Julien Forget, editor, *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, pages 237–246. ACM, 2015. doi:10.1145/2834848.2834850.

16 Joël Goossens. (m, k)-firm constraints and DBP scheduling: Impact of the initial k-sequence and exact feasibility test. In *16th International Conference on Real-Time and Network Systems (RTNS'08)*, pages 61–66, October 2008.

17 Joël Goossens and Christophe Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the 9th International Conference on Real-Time Systems (RTS'01)*, pages 133–148, March 2001.

18 Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignement technique for streams with (m, k)-firm deadlines. *IEEE Trans. Computers*, 44(12):1443–1451, 1995. doi:10.1109/12.477249.

19 E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *6th Real-Time Systems Symposium (RTSS '85), December 3-6, 1985, San Diego, California, USA*, pages 112–122, December 1985.

20 Ning Jia, Ye-Qiong Song, and Françoise Simonot-Lion. Task Handler Based on (m,k)-firm Constraint Model for Managing a Set of Real-Time Controllers. In Nicolas Navet, Françoise Simonot-Lion, and Isabelle Puaut, editors, *15th International Conference on Real-Time and Network Systems - RTNS 2007*, pages 183–194, Nancy, France, 2007. URL: http://hal.inria.fr/inria-00189899.

21 Florian Kluge. `tms-sim` – timing models scheduling simulation framework – release 2014-12. Technical Report 2014-07, University of Augsburg, December 2014. doi:10.13140/2.1.1251.2321.

22 Florian Kluge. Notes on the generation of spin-values for fixed (m, k)-patterns. Technical Report 2016-01, University of Augsburg, January 2016.

23 Florian Kluge, Mike Gerdes, Florian Haas, and Theo Ungerer. A generic timing model for cyber-physical systems. In *Workshop Reconciling Performance and Predictability (RePP'14)*, Grenoble, France, April 2014. doi:10.13140/2.1.1820.4165.

24 Florian Kluge, Florian Haas, Mike Gerdes, and Theo Ungerer. History-cognisant time-utility-functions for scheduling overloaded real-time control systems. In *Proceedings of 7th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2013)*, Sophia Antipolis, France, October 2013.

25 Florian Kluge, Markus Neuerburg, and Theo Ungerer. Utility-based scheduling of (m, k) - firm real-time task sets. In Luís Miguel Pinho, Wolfgang Karl, Albert Cohen, and Uwe Brinkschulte, editors, *Architecture of Computing Systems - ARCS 2015 - 28th International Conference, Porto, Portugal, March 24-27, 2015, Proceedings*, volume 9017 of *Lecture Notes in Computer Science*, pages 201–211. Springer, 2015. doi:10.1007/978-3-319-16086-3_16.

26 Gilad Koren and Dennis E. Shasha. D^{over}; an optimal on-line scheduling algorithm for overloaded real-time systems. In *Proceedings of the Real-Time Systems Symposium - 1992, Phoenix, Arizona, USA, December 1992*, pages 290–299. IEEE Computer Society, 1992. doi:10.1109/REAL.1992.242650.

27 Gilad Koren and Dennis E. Shasha. An approach to handling overloaded systems that allow skips. In *16th IEEE Real-Time Systems Symposium, Palazzo dei Congressi, Via Matteotti, 1, Pisa, Italy, December 4-7, 1995, Proceedings*, pages 110–119. IEEE Computer Society, 1995. doi:10.1109/REAL.1995.495201.

28 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmark for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2015), July 7, 2015, Lund, Sweden*, July 2015.

29 John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*, pages 166–171. IEEE Computer Society, 1989. doi:10.1109/REAL.1989.63567.

30 Peng Li and Binoy Ravindran. Fast, best-effort real-time scheduling algorithms. *IEEE Trans. Computers*, 53(9):1159–1175, 2004. doi:10.1109/TC.2004.61.

31 Peng Li, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *IEEE Trans. Computers*, 55(4):454–469, 2006. doi:10.1109/TC.2006.47.

32 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

33 Carey Douglass Locke. *Best-effort decision-making for real-time scheduling.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1986. URL: http://douglocke.com/Downloads/BE.pdf.

34 Pedro Mejía-Alvarez, Rami G. Melhem, and Daniel Mossé. An incremental approach to scheduling during overloads in real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000), Orlando, Florida, USA, 27-30 November 2000*, pages 283–294. IEEE Computer Society, 2000. doi:10.1109/REAL.2000.896017.

35 Daniel Mossé, Martha E. Pollack, and Yagíl Ronén. Value-density algorithms to handle transient overloads in scheduling. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 278–286. IEEE Computer Society, 1999. doi:10.1109/EMRTS.1999.777475.

36 Enrico Poggi, Yeqiong Song, Anis Koubaa, and Zhi Wang. Matrix-dbp for (m, k)-firm real-time guarantee. In *Real-Time Systems Conference RTS'2003, Paris (France)*, pages 457–482, 2003.

37 Gang Quan and Xiaobo Sharon Hu. Enhanced fixed-priority scheduling with (m, k)-firm guarantee. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000), Orlando, Florida, USA, 27-30 November 2000*, pages 79–88. IEEE Computer Society, 2000. doi:10.1109/REAL.2000.895998.

38 Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):549–559, 1999. doi:10.1109/71.774906.

39 J.-H. Rhu, J.-H. Sun, K. Kim, H. Cho, and J.K. Park. Utility accrual real-time scheduling for (m, k)-firm deadline-constrained streams on multiprocessors. *Electronics Letters*, 47(5):316–317, 2011. doi:10.1049/el.2010.7980.

40 Tiago Semprebom, Carlos Montez, and Francisco Vasques. (m, k)-firm pattern spinning to improve the GTS allocation of periodic messages in IEEE 802.15.4 networks. *EURASIP J. Wireless Comm. and Networking*, 2013:222, 2013. doi:10.1186/1687-1499-2013-222.

41 Sivakumar Swaminathan and Govindarasu Manimaran. A Reliability-Aware Value-Based Scheduler for Dynamic Multiprocessor Real-Time Systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 39–, Washington, DC, USA, 2002. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=645610.661233.

42 Terry Tidwell, Robert Glaubius, Christopher D. Gill, and William D. Smart. Optimizing expected time utility in cyber-physical systems schedulers. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*, pages 193–201. IEEE Computer Society, 2010. doi:10.1109/RTSS.2010.28.

43 Jinggang Wang and Binoy Ravindran. Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis. *IEEE Trans. Parallel Distrib.*

Syst., 15(2):119–133, 2004. doi:10.1109/TPDS.2004.1264796.

44 Richard West and Karsten Schwan. Dynamic window-constrained scheduling for multimedia applications. In *IEEE International Conference on Multimedia Computing and Systems, ICMCS 1999, Florence, Italy, June 7-11, 1999. Volume II*, pages 87–91. IEEE Computer Society, 1999. doi:10.1109/MMCS.1999.778145.

45 Haisang Wu, Binoy Ravindran, E. Douglas Jensen, and Umut Balli. Utility accrual scheduling under arbitrary time/utility functions and multiunit resource constraints. In *in IEEE Real-Time and Embedded Computing Systems and Applications*, pages 80–98, 2004.

A Acronyms

ATS abstract task set
CTS concrete task set
DASA dependent activities scheduling algorithm
DBP distance-based priority
EDF earliest deadline first
GDPA guaranteed dynamic priority assignment
GDPA-S simplified guaranteed dynamic priority assignment
gMUA-MK global multiprocessor utility accrual scheduling algorithm for (m, k)-firm deadline-constrained multimedia streams
GST Goossens' schedulability test
HCUF history-cognisant utility function
LBESA Locke's best-effort scheduling algorithm
MKP evenly distributed (m, k)-patterns
MKP-S evenly distributed (m, k)-patterns with spin values
MKU utility-based scheduling of (m, k)-firm real-time tasks
QoS Quality-of-Service
RM rate monotonic
TUF time-utility function
WCET worst-case execution time

B Additional Results

This appendix contains all performance results that are found during the simulation of task sets with restricted m parameter (see Section 6.2.1), and restricted m parameter and realistic periods (see Section 6.2.3).

B.1 Restricted m Parameter

The diagrams for $r_{\mathrm{m}} = 0.8$ and $r_{\mathrm{m}} = 0.9$ are omitted, as they are already shown in Figures 6 and 7. All other results from Section 6.2.1 are displayed in Figures 12 to 18.

B.2 Restricted m Parameter and Realistic Periods

Figures 19 to 21 show the results from the evaluations discussed in Section 6.2.3.

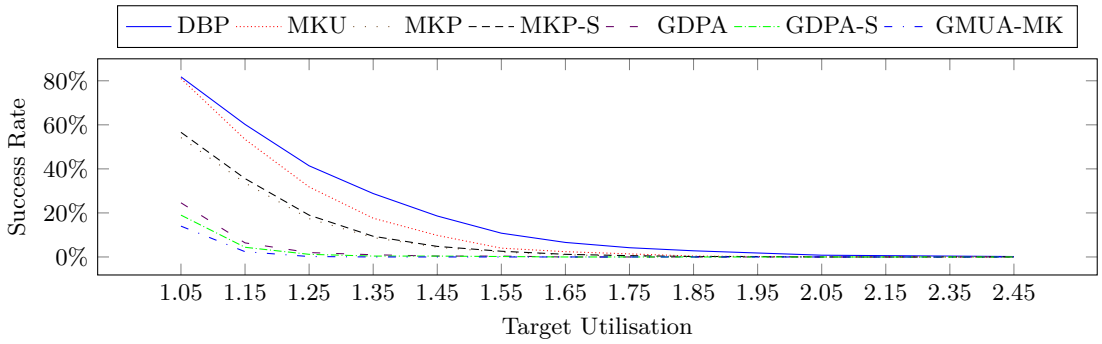


Figure 12 Performance with restricted m_i, $r_{\mathrm{m}} = 0.1$.

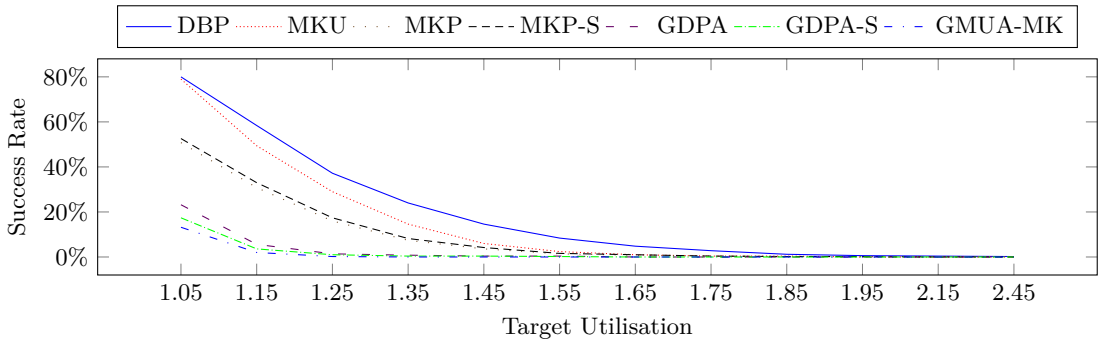


Figure 13 Performance with restricted m_i, $r_{\mathrm{m}} = 0.2$.

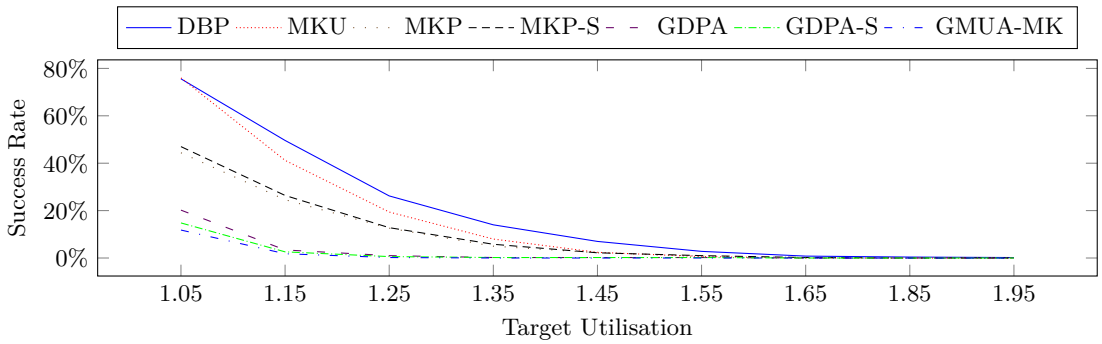


Figure 14 Performance with restricted m_i, $r_{\mathrm{m}} = 0.3$.

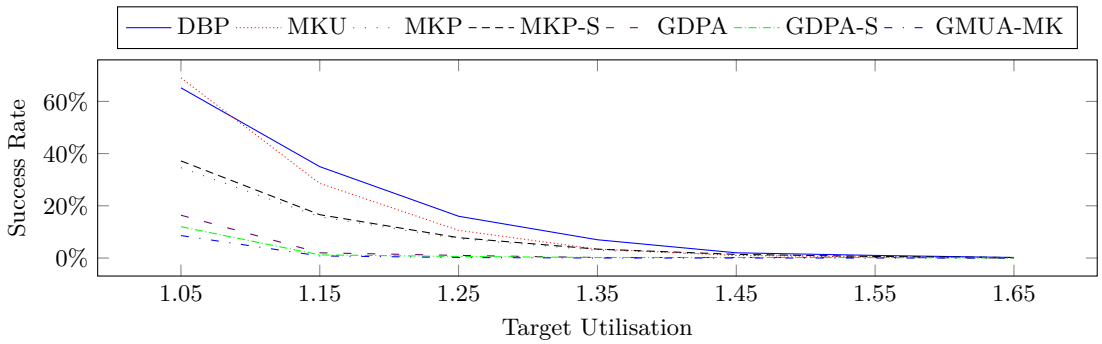


Figure 15 Performance with restricted m_i, $r_{\mathrm{m}} = 0.4$.

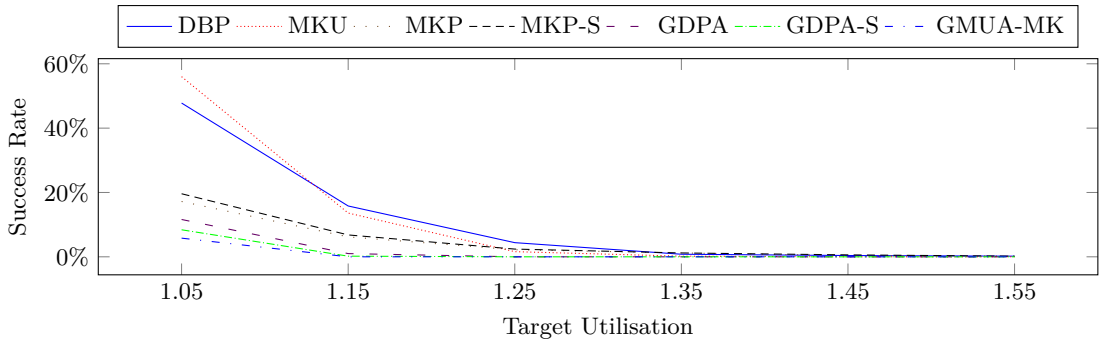


Figure 16 Success rates for $r_{\mathrm{m}} = 0.5$.

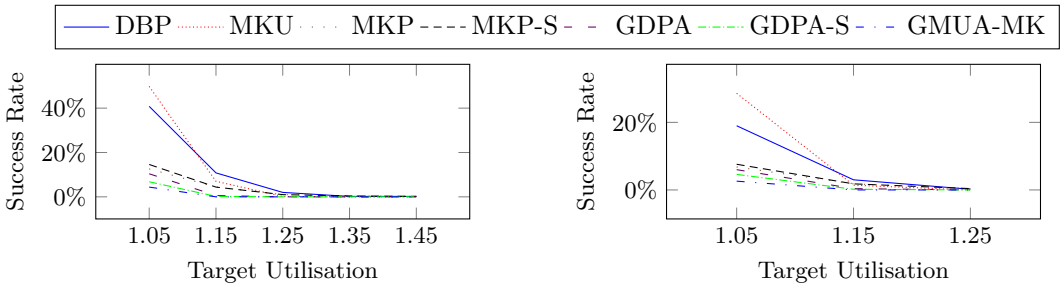


Figure 17 Performance with restricted m_i, $r_{\mathrm{m}} = 0.6$.

Figure 18 Performance with restricted m_i, $r_{\mathrm{m}} = 0.7$.

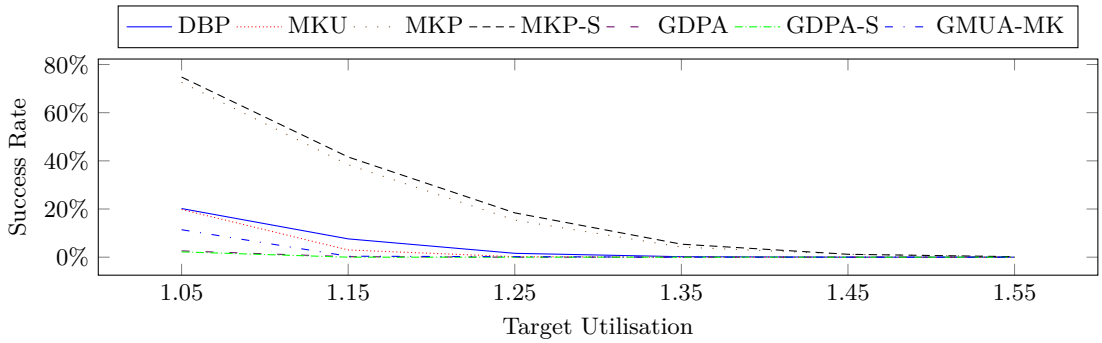


Figure 19 Performance with restricted m_i ($r_{\mathrm{m}} = 0.5$.) and realistic periods.

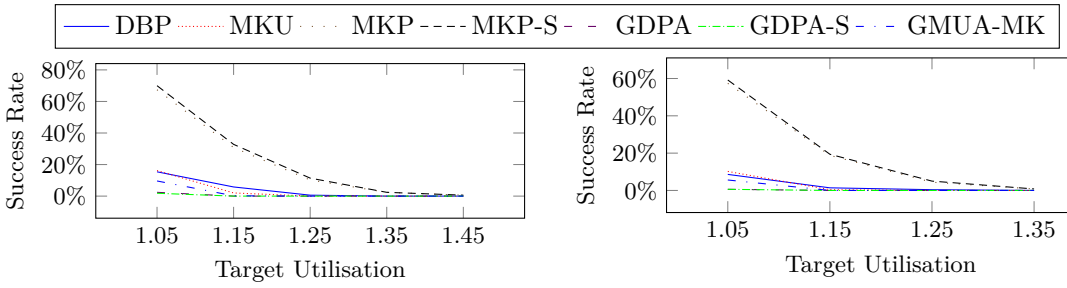


Figure 20 Performance with restricted m_i ($r_{\mathrm{m}} = 0.6$) and realistic periods.

Figure 21 Performance with restricted m_i ($r_{\mathrm{m}} = 0.7$.) and realistic periods.

Quantitative Analysis of Consistency in NoSQL Key-Value Stores*

Si Liu[1], Jatin Ganhotra[2], Muntasir Raihan Rahman[3], Son Nguyen[4], Indranil Gupta[5], and José Meseguer[6]

1 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
siliu3@illinois.edu
2 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
jatin.ganhotra@gmail.com
3 **Microsoft, Redmond, WA, USA**
murahman@microsoft.com
4 **Addepar Inc., Sunnyvale, CA, USA**
son.nguyen@addepar.com
5 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
indy@illinois.edu
6 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
meseguer@illinois.edu

Abstract

The promise of high scalability and availability has prompted many companies to replace traditional relational database management systems (RDBMS) with NoSQL key-value stores. This comes at the cost of relaxed consistency guarantees: key-value stores only guarantee eventual consistency in principle. In practice, however, many key-value stores seem to offer stronger consistency. Quantifying how well consistency properties are met is a non-trivial problem. We address this problem by formally modeling key-value stores as probabilistic systems and quantitatively analyzing their consistency properties by both statistical model checking and implementation evaluation. We present for the first time a formal probabilistic model of Apache Cassandra, a popular NoSQL key-value store, and quantify how much Cassandra achieves various consistency guarantees under various conditions. To validate our model, we evaluate multiple consistency properties using two methods and compare them against each other. The two methods are: (1) an implementation-based evaluation of the source code; and (2) a statistical model checking analysis of our probabilistic model.

2012 ACM Subject Classification Cloud computing, Key-value stores, Model Checking, Rewrite systems
Keywords and Phrases NoSQL Key-value Store, Consistency, Statistical Model Checking, Rewriting Logic, Maude
Digital Object Identifier 10.4230/LITES-v004-i001-a003
Received 2015-12-25 **Accepted** 2016-12-04 **Published** 2016-12-23

Special Issue Editors Javier Campos, Martin Fränzle, and Boudewijn Haverkort
Special Issue Quantitative Evaluation of Systems

* This work was partially supported by NSF CNS 1319527, NSF 1409416, NSF CCF 0964471, and AFOSR/AFRL FA8750-11-2-0084.

1 Introduction

The promise of high scalability and availability has prompted many companies and organizations to replace traditional relational database management systems (RDBMS) with NoSQL key-value stores in order to store large data sets and support an increasing number of users. According to DB-Engines Ranking [16] by September 2016, three NoSQL datastores, namely MongoDB [30], Cassandra [12] and Redis [33], have advanced into the top 10 most popular database engines among 315 systems, highlighting the increasing popularity of NoSQL key-value stores. For example, Cassandra is currently being used at Facebook, Netflix, eBay, GitHub, Instagram, Comcast, and over 1500 more companies.

NoSQL key-value stores invariably replicate application data on multiple servers for greater availability in the presence of failures. Brewer's CAP theorem [11] implies that, under network partitions, a key-value store must choose between consistency (keeping all replicas in sync) and availability (latency). Many key-value stores prefer availability, and thus they provide a relaxed form of consistency guarantees (e.g., eventual consistency [39]). This means key-value store applications can be exposed to stale values. This can negatively impact key-value store user experience. Not surprisingly, in practice many key-value stores seem to offer stronger consistency than they promise. Therefore there is considerable interest in accurately predicting and quantifying what consistency properties a key-value store actually delivers, and in comparing in an objective, and quantifiable way how well properties of interest are met by different designs.

However, the task of accurately predicting such consistency properties is non-trivial. To begin with, building a large scale distributed key-value store is a challenging task. A key-value store usually embodies a large number of components (e.g., membership management, consistent hashing, and so on), and each component can be thought of as source code which embodies many complex design decisions. Today, if a developer wishes to improve the performance of a system (e.g., to improve consistency guarantees, or reduce operation latency) by implementing an alternative design choice for a component, then the only option currently available is to make changes to huge source code bases (e.g., Apache Cassandra [12] has about 345,000 lines of code). Not only does this require many man months, it also comes with a high risk of introducing new bugs, needs understanding in a huge code base before making changes, and is unfortunately not repeatable. Developers can only afford to explore very few design alternatives, which may in the end fail to lead to a better design.

In this paper we address these challenges by proposing a formally model-based methodology for designing and quantitatively analyzing key-value stores. We formally model key-value stores as probabilistic systems specified by *probabilistic rewrite rules* [1], and quantitatively analyze their properties by *statistical model checking* [34, 43]. We demonstrate the practical usefulness of our methodology by developing, to the best of our knowledge for the first time, a formal probabilistic model of Cassandra, as well as of an alternative Cassandra-like design, in Maude [13]. Our formal probabilistic model extends and improves a nondeterministic one we used in [26] to answer *qualitative* binary consistency queries[1] about Cassandra. It models the main components of Cassandra and its environment such as strategies for ordering multiple versions of data and message delay. We have also specified five consistency guarantees that are largely used in industry, *strong consistency* (the strongest consistency guarantee), *causal consistency* (the strongest consistency guarantee that is available in the presence of partitions), *read your writes*, *monotonic reads* and *consistency prefix* (popular intermediate consistency guarantees) [37, 38, 28, 4], in the

[1] A binary consistency query may ask, for example, whether a key-value store read operation is strongly (resp. weakly) consistent or not and other such Boolean-valued questions.

QuaTEx probabilistic temporal logic [1]. Using the PVeStA [3] statistical model checking tool, we have then quantified the satisfaction of such consistency properties in Cassandra under various conditions such as consistency level combination and operation issuing time latency. To illustrate the versatility and ease with which different design alternatives can be modeled and analyzed in our methodology, we have also modeled and analyzed the exact same properties for an alternative Cassandra-like design.

An important question is how much trust can be placed on such models and analysis. That is, how reliable is the predictive power of our proposed methodology? We have been able to answer this question for our case study as follows: (i) we have experimentally evaluated the same consistency properties for both Cassandra and the alternative Cassandra-like design[2]; and (ii) we have compared the results obtained from the formal probabilistic models and the statistical model checking with the experimentally-obtained results. Our analysis indicates that the model-based consistency predictions conform well to consistency evaluations derived experimentally from the real Cassandra deployment, with both showing that Cassandra in fact achieves much higher consistency (sometimes up to strong consistency) than the promised eventual consistency. They also show that the alternative design in most cases is not competitive in terms of the consistency guarantees considered. The purpose of the alternative design was two-fold. First, we wanted to see if the alternative approach would give better accuracy or not. Second, we also wanted to show that it was easier to try the alternative design with our formal model. This is one of the key benefits of building formal models, so that we can quickly try alternative approaches and check their accuracy and performance. Our entire Maude specification, including the alternative design, has less than 1000 lines of code, which further underlines the versatility and ease of use of our methodology at the software engineering level.

Our main contributions include:

- We present a formal methodology for the quantitative analysis of key-value store designs and develop, to the best of our knowledge for the first time, a *formal executable probabilistic model* for the Cassandra key-value store and for an alternative Cassandra-like design.
- We present, to the best of our knowledge for the first time, a statistical model checking analysis for quantifying five consistency guarantees in Cassandra and the alternative design.
- We demonstrate the good predictive power of our methodology by comparing the model-based consistency predictions with experimental evaluations from a real Cassandra deployment on a real cluster. Our results indicate similar consistency trends for the model and the deployment.

This paper extends the results presented in [24, 26] on two substantial ways. First, we explain various consistency models in greater detail including their formal definitions (Section 3) and specifications (Section 5.1). Second, we provide quantitative analysis of consistency using both statistical model checking and implementation-based estimation for three new consistency models (Section 5.2): monotonic reads, consistent pefix and causal consistency [37, 38, 28, 4].

The paper is organized as follows: Section 2 gives a brief overview of Cassandra, Maude, and statistical model checking. Section 3 explains replicated data consistency guarantess in NoSQL datastores. Section 4 presents our probabilistic model of Cassandra and an alternative Cassandra-like design. Section 5 shows the quantitative analysis of consistency by statistical model checking and the implementation-based estimation. Finally, Sections 6 and 7 discuss related work and concluding remarks, respectively.

[2] We implemented the alternative Cassandra-like design by modifying the source code of Apache Cassandra version 1.2.10.

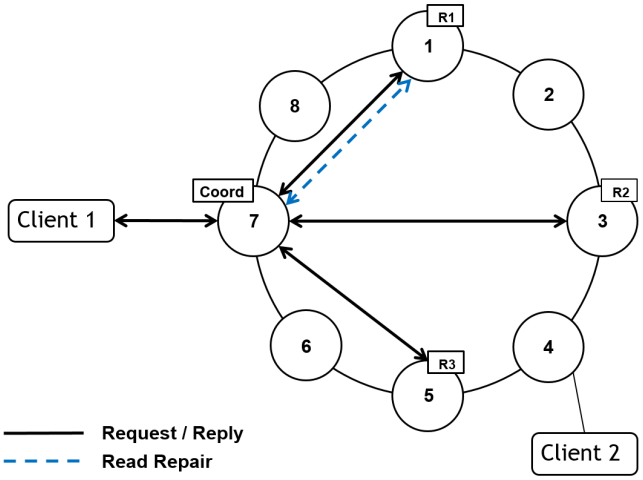


Figure 1 Cassandra deployed in a single cluster of 8 servers with replication factor 3.

2 Preliminaries

2.1 Cassandra Overview

Apache Cassandra [12] is a distributed, scalable, and highly available NoSQL database design. It is distributed over collaborative servers that appear as a single instance to the end client. Data items are dynamically assigned to several servers in the cluster (called the *ring*), and each server (called a *replica*) is responsible for different ranges of the data stored as key-value pairs. Each key-value pair is stored at multiple replicas for fault-tolerance. To place those replicas different strategies can be employed, e.g., the *Simple Strategy* places replicas clockwise in a single data center. For a private cloud Cassandra is typically deployed in a single data center with a single ring structure shared by all its servers.

In Cassandra a client can perform *read* or *write* operations to query or update data. When a client requests a read/write operation to a cluster, the server it is connected to will act as a *coordinator* to forward the request to all replicas that hold copies of the requested key-value pair. According to the specified *consistency level* in the operation, the coordinator will reply to the client with a value/ack after collecting *sufficient* responses from replicas. Cassandra supports tunable consistency levels, with `ONE`, `QUORUM` and `ALL` being the three major ones, meaning that the coordinator will respectively reply with the most recent value (namely, the value with the highest timestamp) to the client after hearing from *one* replica, a *majority* of replicas, or *all* replicas. Thus we call the above strategy of processing reads *Timestamp-based Strategy* (TB). Note that replicas may return different timestamped values to the coordinator. To ensure that all replicas agree on the most recent value, Cassandra uses in the background the *read repair* mechanism to update those replicas holding outdated values.

Figure 1 shows an example Cassandra system deployed in a single data center cluster of eight servers with three replicas and consistency level `QUORUM`. The read/write from client `1` is forwarded to all three replicas `1`, `3` and `5`. The coordinator `7` then replies to client `1` after receiving the first two responses, e.g., from `1` and `3`, to fulfill the request without waiting for the reply from `5`. For a read, upon retrieving all three possibly different versions of values, the coordinator `7` then issues a read repair write with the highest timestamped value to the outdated replica, `1`, in this example. Note that various clients may connect to various coordinators in the cluster, but requests from any client on the same key will be forwarded to the same replicas by those coordinators.

2.2 Rewriting Logic and Maude

Maude [13] is an expressive rewriting-logic-based formal specification language and high-performance simulation and model checking tool for concurrent, object-oriented systems. Maude specifications are executable, and the tool provides a variety of formal analysis methods, including simulation, reachability analysis, and linear temporal logic (LTL) model checking.

A Maude module specifies a *rewrite theory* $(\Sigma, E \cup A, R)$, where:

- Σ is an algebraic *signature*; that is, a set of *sorts*, *subsorts*, and *function symbols*.
- $(\Sigma, E \cup A)$ is a *membership equational logic theory* [13], with E a set of possibly conditional equations and membership axioms, and A a set of equational axioms such as associativity, commutativity, and identity, so that equational deduction is performed *modulo* the axioms A. The theory $(\Sigma, E \cup A)$ specifies the system's states as members of an algebraic data type.
- R is a collection of *labeled conditional rewrite rules* $[l] : t \longrightarrow t'$ **if** *cond*, specifying the system's local transitions.

We briefly summarize the syntax of Maude and refer to [13] for more details. Operators are introduced with the `op` keyword: `op` f `:` $s_1 \ldots s_n$ `->` s. They can have user-definable syntax, with underbars '`_`' marking the argument positions. Equations and rewrite rules are introduced with, respectively, keywords `eq`, or `ceq` for conditional equations, and `rl` and `crl`. The mathematical variables in such statements are declared with the keywords `var` and `vars`, or can be introduced on the fly, in which case they have the form $var{:}sort$. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` ("otherwise") attribute can be applied to a subterm $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied.

A *class* declaration `class` C `|` att_1 `:` s_1`, ...,` att_n `:` s_n declares a class C of objects with attributes att_1 to att_n of sorts s_1 to s_n. An *object instance* of class C is represented as a term `<` O `:` C `|` $att_1 : val_1, ..., att_n : val_n$ `>`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. A *message* is a term of sort `Msg`. A system state is modeled as a term of the sort `Configuration`, and has the structure of a *multiset* made up of objects and messages. The dynamic behavior of a system is axiomatized by specifying each of its transition patterns by a rewrite rule. For example, the rule (with label `l`)

```
rl [l] :  m(O,w)
          < O : C | a1 : x, a2 : O', a3 : z >
       =>
          < O : C | a1 : x + w, a2 : O', a3 : z >
          m'(O',x) .
```

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`, the attribute `a1` of the object `O` is changed to `x + w`, and a new message `m'(O',x)` is generated. Attributes whose values do not change and do not affect the next state, such as `a3` and `a2`, need not be mentioned in a rule. Note that in the above rule `O`, declared as either constant or variable of sort `Oid`, is the object's identifier that can also be a parameter of some function (e.g., the message function `m`).

2.3 Statistical Model Checking and PVeStA

Distributed systems are probabilistic in nature, e.g., network latency such as message delay may follow a certain probability distribution, plus some algorithms may be probabilistic. Systems of this kind can be modeled by *probabilistic rewrite theories* [1] with rules of the form:

$$[l] : t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \;\; if \;\; cond(\overrightarrow{x}) \;\; with \;\; probability \;\; \overrightarrow{y} := \pi(\overrightarrow{x})$$

where the term t' has additional new variables $\overrightarrow{y}$ disjoint from the variables $\overrightarrow{x}$ in the term t. Since for a given matching instance of the variables $\overrightarrow{x}$ there can be many (often infinite) ways to instantiate the extra variables $\overrightarrow{y}$, such a rule is *non-deterministic*. The probabilistic nature of the rule stems from the probability distribution $\pi(\overrightarrow{x})$, which depends on the matching instance of $\overrightarrow{x}$, and governs the probabilistic choice of the instance of $\overrightarrow{y}$ in the result $t'(\overrightarrow{x}, \overrightarrow{y})$ according to $\pi(\overrightarrow{x})$. In this paper we use the above PMaude [1] notation for probabilistic rewrite rules.

Statistical model checking [34, 43] is an attractive formal approach to analyzing probabilistic systems against temporal logic properties. Instead of offering a binary yes/no answer, it provides a quantitative real-valued answer and can verify a property up to a user-specified level of confidence by running Monte-Carlo simulations of the system model. For example, if we consider strong consistency in Cassandra, a statistical model-checking result may be "The Cassandra model satisfies strong consistency 86.87% of the times with 99% confidence". The quantitative answer, however, need not be a percentage or a probability: it may instead be a latency estimation, or a quantitative estimation of some other QoS property. Existing statistical verification techniques assume that the system model is purely probabilistic. Using the methodology in [1, 19] we can eliminate non-determinism in the choice of firing rules. We then use PVeStA [3], an extension and parallelization of the tool VeStA [35], to statistically model check purely probabilistic systems against properties expressed by QuaTEx probabilistic temporal logic [1]. The expected value of a QuaTEx expression is iteratively evaluated w.r.t. two parameters α and δ provided as input by sampling until the size of `(1-`α`)100%` confidence interval is bounded by δ. In this paper we will compute the expected probability of satisfying a property based on definitions of the form `p() = BExp ; eval E[# p()] ;`, where `#`, called "next", is a primitive temporal operator, `BExp` is a consistency-specific predicate (e.g., the predicate `sc?` in Section 5.1.1), and `p()` is a state predicate returning the probabilistic number between 1.0 and 0.0. Informally, the QuaTEx expression consists in a list of definitions of recursive temporal operators such as `p()`, followed by a query of the expected value of a path expression such as `eval E[# p()]` obtained combining the temporal operators.

3 Replicated Data Consistency

Distributed key-value stores usually sacrifice consistency for availability (Brewer's CAP theorem [11]), advocating the notion of weak consistency (e.g., Cassandra promises eventual consistency [12]). However, studies on benchmarking eventually consistent systems have shown that those platforms seem in practice to offer more consistency than they promise [40, 10]. Thus a natural question derived from those observations is "what consistency does your key-value store provide in practice?". We summarize below the prevailing consistency guarantees that have received considerable attention in recent research on distributed data stores [37, 38, 28, 4]. We will focus on five of them (*strong consistency*, *read your writes*, *monotonic reads*, *consistent prefix* and *causal consistency*) in the rest of this paper.

- Strong Consistency (SC) ensures that each read returns the value of the last write that occurred before that read.
- Read Your Writes (RYW) guarantees that the effects of all writes performed by a client are visible to her subsequent reads.
- Monotonic Reads (MR) ensures a client to observe a key-value store increasingly up to date over time.
- Consistent Prefix (CP) guarantees a client to observe an ordered sequence of writes starting with the first write to the system.

- (Time-) Bounded Staleness (BS) restricts the staleness of values returned by reads within a time period.
- Causal Consistency (CC) guarantees that the effects are observed only after their causes: reads will not see a write unless its dependencies are also seen.
- Eventual Consistency (EC) claims that if no new updates are made, eventually all reads will return the last updated value.

Note that SC and EC lie at the two ends of the consistency spectrum, while the other intermediate guarantees are not comparable in general [37].

In [26, 27] we investigated SC, RYW and EC from a *qualitative* perspective using standard model checking, where they were specified using *linear temporal logic* (LTL). The questions we asked and answered there were simply yes/no questions such as "Does Cassandra satisfy strong consistency?" and "In what scenarios does Cassandra violate read your writes?". We indeed showed by counterexamples that Cassandra violates SC and RYW under certain circumstances, e.g., successive write and read with the combinations of lower consistency levels. Regarding EC, the model checking results of our experiments with bounded number of clients, servers and messages conforms to the promise. We refer the reader to [26, 27] for details.

In this paper we look into the consistency issue for Cassandra in terms of SC, RYW, MR, CP and CC from a *quantitative*, statistical model checking perspective. To aid the specification of the five properties (Section 5.1), we now restate them more formally. As all operations from different clients can be totally ordered by their issuing times, we can first view, from a client's perspective, the behavior of a key-value store S as a history $H = o_1, o_2, \ldots, o_n$ of n read/write operations, where any operation o_i can be expressed as $o_i = (k_i, v_i, c_i, t_i)$, where t_i denotes the *global* time when o_i was issued by client c_i, and v_i is the value read from or written to on key k_i and where the sequence is time-increasing, i.e., $t_i \leq t_j$ if $i < j$. We can then define the consistency properties based on H:

- We say S satisfies SC if for any read $o_i = (k, v_i, c_i, t_i)$, provided there exists a write $o_j = (k, v_j, c_j, t_j)$ with $t_j < t_i$, and without any other write $o_h = (k, v_h, c_h, t_h)$ such that $t_j < t_h < t_i$, we have $v_i = v_j$. Note that c_h, c_i and c_j are not necessarily different;
- We say S satisfies RYW if either (1) S satisfies SC, or (2) for any read $o_i = (k, v_i, c_i, t_i)$, provided there exists a write $o_j = (k, v_j, c_j, t_j)$ with $c_i = c_j$ and $t_j < t_i$, and with any other write $o_h = (k, v_h, c_h, t_h)$ such that $c_i \neq c_h$ and $t_j < t_h < t_i$, we have $v_i = v_j$;
- We say S satisfies MR if for any two reads $o_{ri} = (k_{ri}, v_{ri}, c_{ri}, t_{ri})$ and $o_{rj} = (k_{rj}, v_{rj}, c_{rj}, t_{rj})$ in the sequence of reads $o_{r1}, o_{r2}, \ldots, o_{rn}$, provided there exists a sequence of writes $o_{w1}, o_{w2}, \ldots, o_{wm}$, if $v_{ri} = v_{wg}$ and $v_{rj} = v_{wh}$, we have $t_{wg} \leq t_{wh}$, provided $k_{ri} = k_{wg}$ and $k_{rj} = k_{wh}$. Note that c_{ri}, c_{rj}, c_{wg} and c_{wh} are not necessarily different;
- We say S satisfies CP if for a sequence of reads $o_{r1}, o_{r2}, \ldots, o_{rm}$, provided there exists a sequence of writes $o_{w1}, o_{w2}, \ldots, o_{wn}$, we have, if $n \geqslant m$, $v_{r1} = v_{w1}, v_{r2} = v_{w2}, \ldots, v_{rm} = v_{wm}$, provided that $k_{r1} = k_{w1}, k_{r2} = k_{w2}, \ldots, k_{rm} = k_{wm}$; otherwise, $v_{r1} = v_{w1}, v_{r2} = v_{w2}, \ldots, v_{rn} = v_{wn}, v_{rn+1} = v_{wn}, \ldots, v_{rm} = v_{wn}$, provided $k_{r1} = k_{w1}, k_{r2} = k_{w2}, \ldots, k_{rn} = k_{wn}, k_{rn+1} = k_{wn}, \ldots, k_{rm} = k_{wn}$. Note that c_{ri} and c_{wj} are not necessarily different, and the key pairs (e.g., k_{ri} and k_{rj}) are not necessarily the same;
- We introduce two notions, *causality order* and *serialization* [2, 38], before defining S satisfying CC. We say $o_i = (k_i, v_i, c_i, t_i)$ is *causally ordered* before $o_j = (k_j, v_j, c_j, t_j)$ if one of the following three cases holds: (i) $c_i = c_j$ and $t_i < t_j$; (ii) $v_j = v_i$, provided o_i is a write while o_j a read; (iii) there exists some other o_k such that o_i is causally ordered before o_k that is causally ordered before o_j (i.e., transitivity). We say L is a *serialization* of a history H if L is a linear sequence containing exactly the operations of H such that L satisfies SC. We say a serialization L *respects* a causality order if, for any two operations o_i and o_j in L, o_i is causally

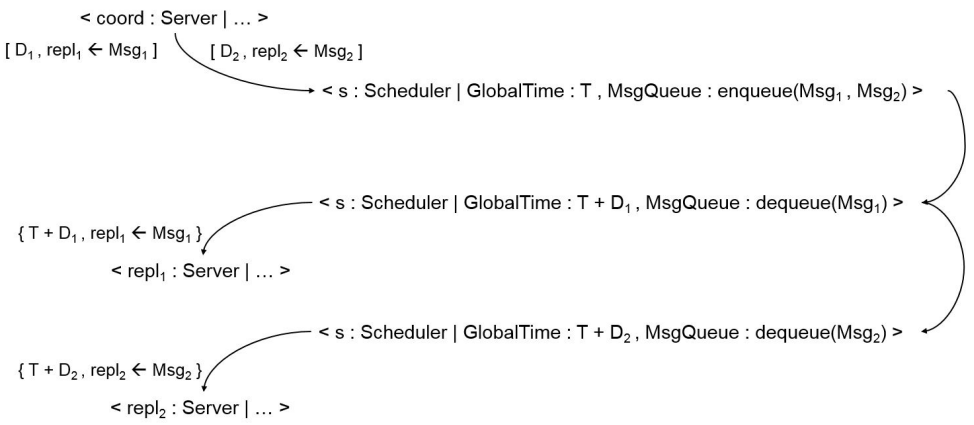


Figure 2 Visualization of rewrite rules for forwarding requests from a coordinator to the replicas.

ordered before o_j implies o_i precedes o_j in L. We then say S satisfies CC if H has a causality order such that for each client c_i there is a serialization of all operations from c_i and all writes (probably from different clients) in H that respects that causality order.

4 Probabilistic Modeling of Cassandra Designs

This section describes a formal probabilistic model of Cassandra as well as an alternative Cassandra-like design. Section 4.1 shows the underlying communication model of Cassandra components, and how the associated non-deterministic rewrite rules are transformed into purely probabilistic ones in Maude. Section 4.2 presents an alternative Cassandra-like design in terms of read processing strategy. The entire executable Maude specifications are available at `https://sites.google.com/site/siliunobi/lites-cassandra`.

4.1 Formalizing Probabilistic Communication in Cassandra

In [26] we built a formal executable model of Cassandra summarized in Section 2.1. Specifically, we modeled the ring structure, clients and servers, messages, and Cassandra's dynamics. Moreover, we also introduced a *scheduler* object to schedule messages by maintaining a global clock `GlobalTime`[3] and a queue of inactive/scheduled messages `MsgQueue`. By activating those messages, it provides a deterministic total ordering of messages[4] and allows synchronization of all clients and servers, aiding formal analysis of consistency properties (Section 5.1).

To illustrate the underlying communication model, Figure 2 visualizes a segment of the system transitions showing how messages flow between a coordinator and the replicas through the scheduler in terms of rewrite rules. The delayed messages (of the form `[...]`) `[D1, repl1 <- Msg1]` and `[D2, repl2 <- Msg2]`, targeting replicas `repl1` and `repl2`, are produced by the coordinator at global time `T` with the respective message delays `D1` and `D2`. The scheduler then enqueues both messages for scheduling. As the global time advances, messages eventually become active (of the

[3] Although in reality synchronization can never be exactly reached due to clock skew [23], cloud system providers use NTP or even expensive GPS devices to keep all clocks synchronized (e.g., Google Spanner [15]). Thus our abstraction of a global clock is reasonable.

[4] It is possible to have two active messages with the same delivery time. In that case messages will be ordered alphabetically.

form {...}), and are appropriately delivered to the replicas. For example, the scheduler first dequeues `Msg1` and then `Msg2` at global time `T + D1` and `T + D2` respectively, assuming `D1 < D2`. Note that messages can be consumed by the targets only when they are active.

As mentioned in Section 2.3, we need to eliminate nondeterminism in our previous Cassandra model prior to statistical model checking. This can be done by transforming nondeterministic rewrite rules to purely probabilistic ones. Below we show an example transformation, where both rules illustrate how the coordinator reacts upon receiving a read reply `ReadReplySS` from a replica, with `KV` the returned key-value pair of the form `(key,value,timestamp)`, `ID` and `A` the read and client's identifiers, and `CL` the read's consistency level, respectively. The coordinator `S` adds `KV` to its local buffer, and returns to `A` the highest timestamped value determined by `tb` via the message `ReadReplyCS`, provided it has collected the consistency-level number of responses determined by `cl?`.

In the nondeterministic version `[...-nondet]`, the outgoing message is equipped with a delay `D` nondeterministically selected from the delay set `delays` where `DS` defines the rest delays of the set. We keep the set unchanged so that standard model checking will explore all possible choices of delays each time the rule is fired. For example, if `delays` are : `(2.0,4.0)`, two read replies will be generated nondeterministically with the delays `2.0` and `4.0` time units respectively, each of which will lead to an execution path during the state space exploration. Note that `AS` refers to the rest of the attributes of server `S`.

```
crl [on-rec-rrep-coord-nondet] :
    {T, S <- ReadReplySS(ID,KV,CL,A)}
    < S : Server | buffer: BF, delays: (D,DS), AS >
 =>
    < S : Server | buffer: BF', delays: (D,DS), AS >
    (if cl?(CL,BF') then
     [D, A <- ReadReplyCS(ID,tb(BF'))]
     else none fi)
 if BF' := add(ID,KV,BF) .
```

We transform the above rule to the probabilistic version `[...-prob]`, where the delay `D` is distributed according to the parameterized probability distribution function `distr(...)`. Once the rule fires, only one read reply will be generated with a probabilistic real-valued message delay.

```
crl [on-rec-rrep-coord-prob] :
    {T, S <- ReadReplySS(ID,KV,CL,A)}
    < S : Server | buffer: BF, AS >
 =>
    < S : Server | buffer: BF', AS >
    (if cl?(CL,BF') then
     [D, A <- ReadReplyCS(ID,tb(BF'))]
     else none fi)
 if BF' := add(ID,KV,BF) with probability D := distr(...) .
```

Likewise, all nondeterministic rules in our previous model can be transformed to purely probabilistic rewrite rules. Furthermore, as explained in [1, 19], the use of continuous time and the actor-like nature of the specification ensure that *at most one probabilistic rule will be enabled at each time instant*, thus eliminating any remaining nondeterminism from the firing of rules.

4.2 Alternative Strategy Design

Two major advantages of our model-based approach are: (1) the ease of exploring different design alternatives (e.g., designing an alternative strategy to TB described in Section 2.1) in early design stages, and (2) the ability to predict their effects before implementation. Here we illustrates the first part by presenting as an alternative design the *Timestamp-agnostic Strategy* (TA). The key idea is that, instead of using timestamps to decide which value will be returned to the client as TB does (Section 2.1), TA uses the values themselves to decide which replica has the latest value. For example, if the replication factor is 3, then for a `QUORUM` read, the coordinator checks whether the values returned by the first two replicas are identical: if they are, the coordinator returns that value; otherwise it waits for the third replica to return a value. If the third value matches one of the first two values, the coordinator returns the third value. So for a `QUORUM` read TA guarantees that the coordinator will reply with the value that has been stored at a majority of replicas. For an `ALL` read, the coordinator compares all three values; if they are all the same, it returns that value. Notice that TA and TB agree on processing `ONE` reads.

To formalize TA (or other alternative strategies) we only need to specify the corresponding functions of the returned values from the replicas buffered at the coordinator, as we defined `tb` for TB, and for deciding if enough responses have been fetched by the coordinator, as we defined `cl?` for TB, without redefining the underlying model. Note that our component-based model also makes it possible to dynamically choose the optimal strategy in favor of consistency guarantees. More precisely, once the system has been specified endowed with a family of strategies having respective strengths in consistency guarantees (which can be measured by statistical model checking), the coordinator can invoke the corresponding strategy-specific function based on the client's specified preference. For example, given strategies S1/S2 offering consistency properties C1/C2, if a client issues two consecutive reads with desired consistency C1, C2, respectively, the coordinator will generate, e.g., the C1-consistent value for the preceding read, by calling the strategy function for S1.

On the other hand, the key change for implementing TA is to implement our own `Resolve()` function for class `RowDataResolver`. We rewrote Cassandra's `resolveSupersetNew()` function for the TA approach. This took some code modification, and it required a complete understanding of the Cassandra internal code details. In terms of Cassandra code changes, 150 lines of code were added and 50 were removed. We observe that formalizing TA in Maude was much easier and faster than implementing it in Cassandra.

5 Quantitative Analysis of Consistency in Cassandra

How well do our Cassandra model and its TA alternative design satisfy different consistency guarantees? Does TA provide better consistency than TB based on our model? Are those results in agreement with actual implementations? We propose to investigate these questions by statistical model checking and by implementation-based evaluation of those consistency properties in terms of the two strategies.

5.1 Formalization of Consistency Properties

5.1.1 Strong Consistency

Scenario. SC ensures that each read returns the value of the last write that occurred before that read. Thus we design a scenario with one read and two writes to see if the read will return the

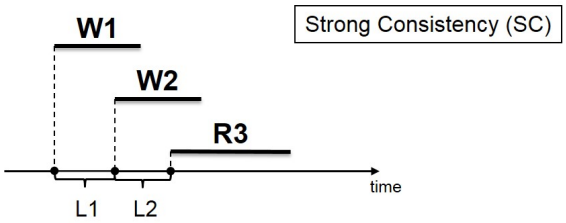


Figure 3 Experimental Scenario of Statistical Model Checking of SC.

last write[5]. Figure 3 shows the experimental scenario for SC, with each parallel line denoting one session of one client. The scenario consists of three consecutive operations, W1, W2 and R3, issued by three different clients, respectively, where `L1` and `L2` are the issuing latencies between them. We choose to experiment with consistency level `ONE` for both W1 and W2 to evaluate different consistency levels for R3. Thus we name each subscenario (TB/TA-O/Q/A) depending on the target strategy and R3's consistency level, e.g., (TB-Q) refers to the case checking SC for TB with R3 of `QUORUM`.

Formal Specification of SC. Based on the consistency definition (Section 3) and the above scenario, SC is satisfied if R3 reads the value of W2. Thus we define a parameterized predicate `sc?(A,A',C)` that holds if we can match the value returned by the subsequent read `O` (R3 in this case) from client `A` with that in the preceding write `O'` (W2 in this case) from client `A'`. Note that the attribute `store` *lists* the associated information of each operation issued by the client, e.g., operation `O` was issued at global time `T` on key `K` with returned/written value `V` for a read/write. Note that in the rest of this paper `REST` refers to the rest objects of the entire configuration (Section 2.2).

```
op sc? : Address Address Config -> Bool .

eq sc?(A,A',< A  : Client | store : ((O, K,V,T ), ...), ... >
            < A' : Client | store : ((O',K,V,T'), ...), ... > REST) = true .

eq sc?(A,A',C) = false [owise] .
```

5.1.2 Read Your Writes

Scenario. RYW guarantees that the effects of all writes performed by a client are visible to her subsequent reads. Thus we design a scenario with two writes and one read from the same client to see if the read will return the last write; we also add another write from a different client since returning the latest write from a different client will also satisfy RYW. Figure 4 shows the experimental scenario for RYW. The scenario consists of four operations, where W1, W2 and R3 are issued by one client and *strictly ordered* (a subsequent operation will be blocked until the preceding one finishes) while W4 is from the other client[6]. The issuing latency `L` is tunable, which

[5] Note that we need to minimize all the experimental scenarios for the statistical model checking in this paper for two reasons: (1) statistical model checking can only support a limited number of operations to avoid state space explosion; and (2) we observed that in our experiments adding a few more operations would not affect the results much but would aggravate the statistical model checking.

[6] Section 3 describes two disjoint cases for RYW, which we mimic with tuneable `L`: if `t4 < t2`, only W2 is RYW-consistent; otherwise both W2 and W4 are RYW-consistent.

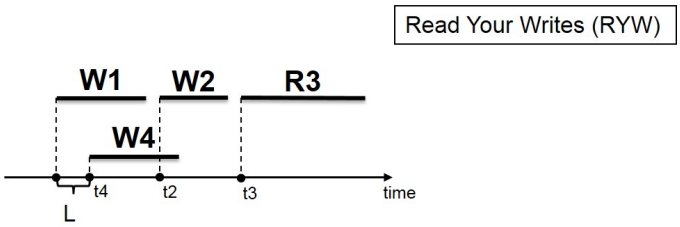


Figure 4 Experimental Scenario of Statistical Model Checking of RYW.

can vary the issuing time of W4. Thus we can derive the corresponding cases in RYW's definition (Section 3), and specify and analyze the property accordingly. We choose to experiment with consistency level `ONE` for both W1 and W4 to evaluate different combinations of consistency levels for W2 and R3. The only possible cases violating RYW are, if we forget W4 for the moment, (R3,W2) = (O,O)/(O,Q)/(Q,O) due to the fact that a read is guaranteed to see its preceding write from the same client, if `R + W > RF` with `R` and `W` the respective consistency levels and `RF` the replication factor. For example, if the replication factor `RF` is 3 (thus `ONE/QUORUM/ALL` is 1/2/3), a `ONE` read `R` and a `QUORUM` write `W` cannot ensure RYW.

Formal Specification of RYW. We define a parameterized predicate `ryw?(A,A',C)` that holds if the subsequent read `O2` (R3 in this case) returns the appropriate value required for RYW to hold. This can happen in two different ways: (1) `O2` returns the value `V` by the preceding write `O1` (W2 in this case), or (2) `O2` returns the value `V'` by a more recent write `O3` (W4 in this case if issued after W2, which is determined by `T3 >= T1`).

```
op ryw? : Address Address Config -> Bool .

eq ryw?(A,A',< A  : Client | store : (..., (O1,K,V,T1), (O2,K,V,T2), ...),
     ... > REST) = true .   --- case (1)

ceq ryw?(A,A',< A  : Client | store : (..., (O1,K,V,T1), (O2,K,V',T2), ...), ... >
        < A' : Client | store : ((O3,K,V',T3), ...), ... > REST) = true
     if T3 >= T1 .   --- case (2)

eq ryw?(A,A',C) = false [owise] .
```

5.1.3 Monotonic Reads

Scenario. MR ensures that a client will observe a key-value store increasingly up to date over time. Thus we design a scenario with two writes followed by two consecutive reads to see if the two reads will return the two writes in order. Figure 5 shows the experimental scenario for MR. We consider a scenario with four operations, where two writes, W1 and W2, are issued by two different clients, and a third client issues two strictly ordered reads, R3 and R4. `L1` and `L2` are the issuing latencies between W1 and W2, and W2 and R3, respectively. We choose to experiment with the same consistency level for R3 and R4, and with consistency level `ONE` for W1 to evaluate different combinations of consistency levels for W2 and R3/R4.

Formal Specification of MR. We define a parameterized predicate `mr?(A1,A2,A3,C)` that holds if the subsequent read `O4` (R4 in this case) from client `A3` returns the appropriate value required for MR to hold. This can happen in three different ways: (1) if the preceding read `O3` (R3 in this case) gets the default value *dft*, any value returned by `O4` is MR-consistent; (2) if `O3` reads the

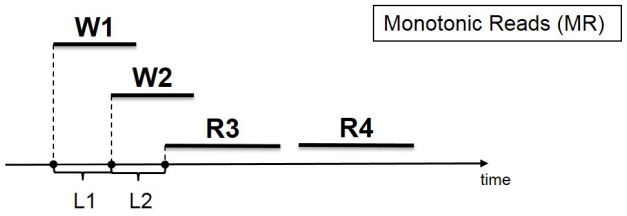


Figure 5 Experimental Scenario of Statistical Model Checking of MR.

value V1 from client A1's write O1 (W1 in this case), O4 has to return the value V1 or V2 (from client A2's write O2) to satisfy MR; and (3) if O3 returns V2, O4 needs to match it to guarantee MR.

```
op mr? : Address Address Address Config -> Bool .

eq mr?(A1,A2,A3,< A3 : Client | store: ((O3,K,dft,T3), (O4,K,V4,T4), ...),
     ... > REST) = true .    --- case (1)

eq mr?(A1,A2,A3,< A1 : Client | store: ((O1,K,V1,T1), ...), ... >
                < A3 : Client | store: ((O3,K,V1,T3), (O4,K,V1,T4), ...),
     ... > REST) = true .
eq mr?(A1,A2,A3,< A1 : Client | store: ((O1,K,V1,T1), ...), ... >
                < A2 : Client | store: ((O2,K,V2,T2), ...), ... >
                < A3 : Client | store: ((O3,K,V1,T3), (O4,K,V2,T4), ...),
     ... > REST) = true .    --- case (2)

eq mr?(A1,A2,A3,< A2 : Client | store: ((O2,K,V2,T2), ...), ... >
                < A3 : Client | store: ((O3,K,V2,T3), (O4,K,V2,T4), ...),
     ... > REST) = true .    --- case (3)

eq mr?(A1,A2,A3,C) = false [owise] .
```

5.1.4 Consistent Prefix

Scenario. CP guarantees that a client will observe an ordered sequence of writes starting with the first write to the system. Thus we design a scenario with six operations on two different keys[7], where four strictly ordered writes, W1, W2, W3 and W4, are interleaved on two keys, K1 and K2, and issued by a single client, and two strictly ordered reads, R1 (on K1) and R2 (on K2), by the other client. Figure 6 shows the experimental scenario for CP. The issuing latency L between W1 and R1 is tunable, which can vary the issuing time of R1. We choose to experiment with the same consistency level for the writes/reads to evaluate different combinations of consistency levels.

Formal Specification of CP. We define a parameterized predicate cp?(A1,A2,C) that holds if the successive reads O5 (R5 in this case) and O6 (R6 in this case) from client A2 return an ordered sequence of writes. This can happen in three different ways: (1) if O5 gets the default value *dft1* for K1, O6 can only read the default value *dft2* for K2 to satisfy CP; (2) if O5 returns the value V1 from client A1's write O1 (W1 in this case), O6 has to return the value V2 or *dft2* (either case

[7] For reads on a single key in a system like Cassandra where writes completely overwrite previous values of a key, eventual consistency reads can guarantee consistent prefix. So we consider reads on multiple keys [37].

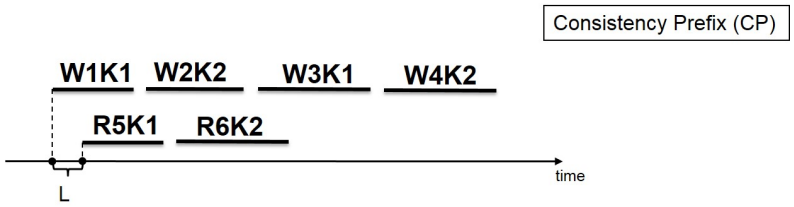


Figure 6 Experimental Scenario of Statistical Model Checking of CP.

guarantees an ordered sequence of writes); and (3) if O5 reads the value V3 from the write O3 (W3 in this case), both V2 and V4 are CP-consistent.

```
op cp? : Address Address Config -> Bool .

eq cp?(A1,A2,< A2 : Client | store: ((O5,K1,dft1,T5),
     (O6,K2,dft2,T6), ...), ... > REST) = true .    --- case (1)

eq cp?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), ...), ... >
             < A2 : Client | store: ((O5,K1,V1,T5), (O6,K2,dft2,T6),
     ...), ... > REST) = true .
eq cp?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O5,K1,V1,T5), (O6,K2,V2,T6),
     ...), ... > REST) = true .    --- case (2)

eq cp?(A1,A2,< A1 : Client | store: (..., (O2,K2,V2,T2), (O3,K1,V3,T3), ...), ... >
             < A2 : Client | store: ((O5,K1,V3,T5), (O6,K2,V2,T6),
     ...), ... > REST) = true .
eq cp?(A1,A2,< A1 : Client | store: (..., (O3,K1,V3,T3), (O4,K2,V4,T4), ...), ... >
             < A2 : Client | store: ((O5,K1,V3,T5), (O6,K2,V4,T6),
     ...), ... > REST) = true .    --- case (3)

eq cp?(A1,A2,C) = false [owise] .
```

5.1.5 Causal Consistency

Scenario. CC guarantees that the effects are observed only after their causes: reads will not see a write unless its dependencies are also seen. Thus we design a scenario with a write and a read from two different clients to bridge them by causality. Figure 7 shows the experimental scenario for CC. We consider a scenario with five operations on two different keys, where client C1 issues two strictly ordered writes, W1 and W2, on the keys, K1 and K2, respectively, and the other client C2 first issues a read on K2, and then issues two consecutive operations W4 and R5 on K1. Note that, although operations from the same session of a client are causally ordered (e.g., W1 and W2), we check in this scenario causality that arises when the first read R3 returns the value of W2, establishing the causality between W2 and R3, and thus W1 and W4 (i.e., corresponding to the definition of CC (Section 3), we check from client C2's perspective if there is a serialization of all five operations that respects the causality order). The issuing latency L between W1 and R3 is tunable, which can vary the issuing time of R3. We choose to experiment with the same consistency level ONE for W1, W2 and R3 to evaluate different combinations of consistency levels for W4 and R5.

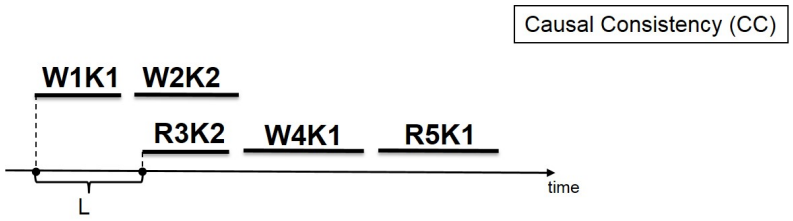


Figure 7 Experimental Scenario of Statistical Model Checking of CC.

Formal Specification of CC. We define for CC a parameterized predicate `cc?(A1,A2,C)` that holds if `O5` (R5 in this case) returns the value `V4` from client `A2`'s write `O4` (W4 in this case), provided `O3` (R3 in this case) reads the value `V2` from client `A1`'s write `O2` (W2 in this case); otherwise, any value returned by `O5` is CC-consistent, since no causality arises.

```
op cc? : Address Address Config -> Bool .

eq cc?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O3,K2,V2,T3), (O4,K1,V4,T4), (O5,K1,V4,T5),
     ...), ... > REST) = true .   --- causality arises

eq cc?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O3,K2,dft2,T3), (O4,K1,V4,T4), (O5,K1,V5,T5),
     ...), ... > REST) = true .   --- no causality

eq cc?(A1,A2,C) = false [owise] .
```

5.2 Analysis Results for Consistency Guarantees

We are now ready to present the analysis results for all those consistency models on both statistical model checking and implementation-based evaluation. We compare the two strategies, TA and TB, from two aspects: (1) if both show the similar trends as the latency increases; and (2) if one provides better consistency than the other.

First, we define the following setting for our experimental scenarios of statistical model checking:

- We consider a single cluster of 4 servers, and the replication factor of 3.
- All replicas are initialized with default key-value pairs.
- Each read/write can have consistency level `ONE`, `QUORUM` or `ALL`.
- We consider the lognormal distribution for message delay with the mean μ `= 0.0` and standard deviation σ `= 1.0` [9].
- All consistency probabilities are computed with a 99% confidence level of size at most 0.01 (Section 2.3).

Note that for the rest of this paper we name each subscenario (TB/TA-OO/OQ/QO/...) depending on the target strategy and the combination of consistency levels for the experimental scenarios regarding the consistency models excluding SC. For simplicity, we let an operation occur immediately upon its preceding operation from the same session (or client) finishes.

Regarding the experimental setup on the other hand, we deploy Cassandra on a single cluster of 4 Emulab [41] servers within the same rack.[8] The servers used were Dell r710 2U servers with

[8] For a private cloud, Cassandra is typically deployed in a single data center with a single ring structure shared by all its servers. The network latency effects will not impact our experiment results as all servers are in

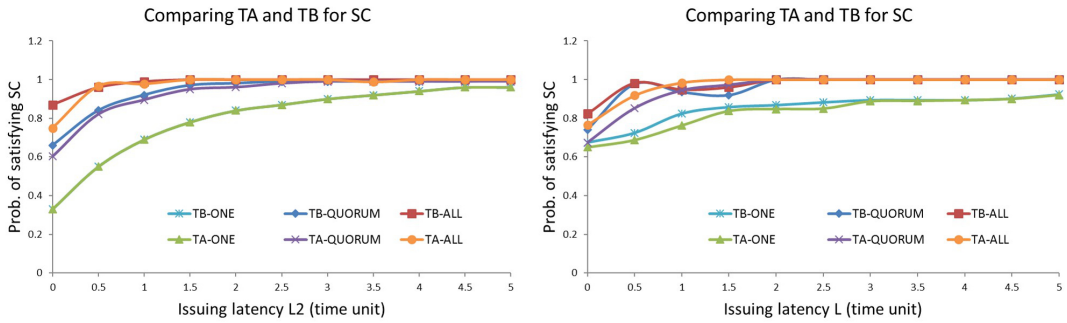


Figure 8 Probabilities of Satisfying SC by Statistical Model Checking (Left) and by Real Deployment Run (Right).

the following configuration: one 2.4 GHz 64-bit Quad Core Xeon E5530 Nehalem processor, 5.86 GT/s bus speed, 8 MB L3 cache, VT (VT-x, EPT and VT-d) support and 12 GB 1066 MHz DDR2 RAM; the OS used was 64-bit Ubuntu 14.04 LTS, 3.13.0 kernel and Cassandra version was 1.2.10. We use YCSB [14] to inject read/write workloads. For RYW, MR, CP and CC tests, we use two separate YCSB clients.[9] Our test workloads are read-heavy (which are representative of many real-world workloads such as Facebook's photo storage [8]) with 90% reads, and we vary consistency levels between `ONE`, `QUORUM`, and `ALL`. We run Cassandra and YCSB clients for fixed time intervals (each client performs 20,000 operations) and log the results. Based on the logs generated, we calculate the percentage of reads that satisfy various consistency models considered in this paper. We perform binning on all the results to generate the issuing latency intervals. The reason is that in our statistical model checking experiments we can specify the exact latency interval between operations, but the same is impossible for a real deployment. From our real deployment experiments, we get the probabilities for a continuous range of latency values, instead of a discrete set as in the statistical model checking experiments. Thus we divide all the results and perform binning to generate the issuing latency intervals. Note that other experimental configurations such as the replication factor, consistency levels and message delay distribution follow our setup for statistical model checking.

5.2.1 Analysis Results for SC

The left plot in Figure 8 shows the resulting probability of satisfying SC by statistical model checking, where the probability (of R3 reading the value of W2) is plotted against the issuing latency (`L2`) between them. Regarding TB, from the results (and intuitively), given the same issuing latency, increasing the consistency level provides higher consistency; given the same consistency level, higher issuing latency results in higher consistency (as the replicas converge, a sufficiently later read (R3) will return the consistent value up to 100%). Surprisingly, `QUORUM` and `ALL` reads start to achieve SC within a very short latency around `0.5` and `1.5` time units respectively (with `5` time units for even `ONE` reads).

On the other hand, all observations for TB apply to TA in general. In fact, for `QUORUM` and `ALL` reads, the two strategies perform almost the same, except that: (1) for `ALL` reads, TB provides

the same rack and the time difference between each pair of requests would be in the same range. While showing scalability is not the goal of this paper, we wish to do larger scale experiments in the future. There are resource challenges related to scaling the model-checking to larger scales (e.g., parallelizing it the right way), and we hope to solve this in our future work.

[9] Even if the tests allow a large number of YCSB clients, we set it to 2 to match our statistical model checking experimental setting.

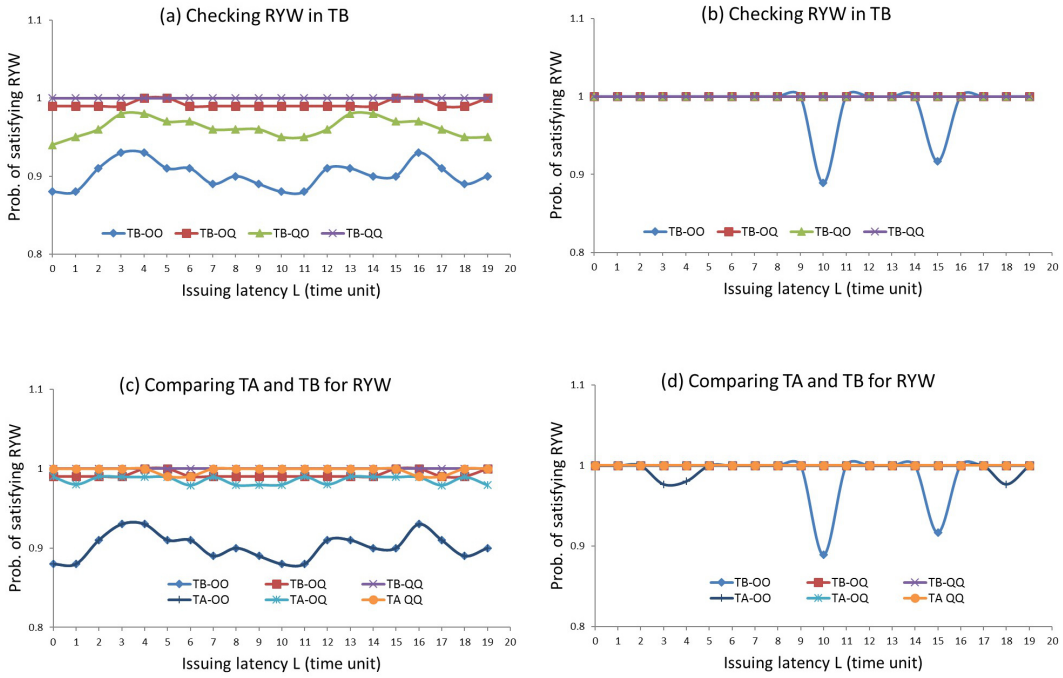


Figure 9 Probabilities of Satisfying RYW by Statistical Model Checking (Left) and by Real Deployment Run (Right).

noticeably more consistency than TA within an extremely short latency of `0.5` time units; and (2) for `QUORUM` reads, TB offers slightly more consistency than TA within `2.5` time units.

Based on these results it seems fair to say that both TB and TA provide high SC, especially with `QUORUM` and `ALL` reads. The consistency difference between the two strategies results from the overlap of R3 and W2. More precisely, since the subsequent read has higher chance to read multiple versions of the key-value pair with lower issuing latency, TA, only relying on the version itself, will return the matched value that is probably stale.

Regarding the implementation-based evaluation (the right plot in Figure 8), we show the resulting, experimentally computed probability of strongly consistent reads against `L2` (Figure 3) for deployment runs regarding the two strategies (only for `QUORUM` and `ALL` reads). Overall, the results indicate similar trends for the model predictions and real deployment runs (both plots in Figure 8): for example, for both model predictions and deployment runs, the probability is higher for `ALL` reads than for `QUORUM` reads regarding both strategies, especially when `L2` is low; consistency does not vary much with different strategies. Note that we observe the dips that break the monotonic behavior of TB-ALL and TB-QUORUM at a lower value of `L` because it is possible that the updated value from W2 has not reached all the replicas. With a higher value of `L`, we do not observe any break from the monotonic behavior.

5.2.2 Analysis Results for RYW

Figure 9(a) shows the resulting probability of satisfying RYW, where the probability (of R3 reading the value of W2 or a more recent value) is plotted against the issuing latency (`L`) between W1 and W4. From these results it is straightforward to see that scenarios (TB-OA/QQ/AO/AA) guarantee RYW due to the fact "`R3 + W2 > RF`". Since we have already seen that the Cassandra model satisfied SC quite well, it is also reasonable that all combinations of consistency levels

provide high RYW consistency, even with the lowest combination (O,O) that already achieves a probability around 90%. Surprisingly, it appears that a QUORUM read offers RYW consistency nearly 100%, even after a preceding write with its consistency level down to ONE (scenario (TB-OQ)). Another observation is that, in spite of the concurrent write from the other client, the probability of satisfying RYW stays fairly stable.

Figure 9(c) shows the comparison of TA and TB regarding RYW, where for simplicity we only list three combinations of consistency levels from R3's perspective with W2's consistency level fixed to ONE (in fact, with W2's consistency level increases, the corresponding scenarios will provide even higher consistency). In general, all observations for TB apply to TA, and it seems fair to say that both TA and TB offer high RYW consistency. The two strategies agree on the combination (O,O). However, TA cannot offer higher consistency than TB in any other scenario, with TA providing slightly lower consistency for some points, even though TA's overall performance is close to TB's over issuing latency. One reason is that TA does not respect the fact "R + W > RF" in general (e.g., two strictly ordered Quorum write and read cannot guarantee RYW).

Regarding the implementation-based evaluation (Figure 9(b) and (d)), we show the resulting probability of RYW-consistent reads against L (Figure 4) for deployment runs regarding two strategies. Both the model predictions and deployment runs show very high probability of satisfying RYW. This is expected since for each client the operations are mostly ordered, and for any read operation from a client, we expect any previous write from the same client to be committed to all replicas. For the deployment runs, we observe that we get 100% RYW consistency, except for scenario (TB-OO), which matches expectations, since ONE is the lowest consistency level and does not guarantee anything more than EC. This also matches our model predictions in Figure 9, where we see that the probability of satisfying RYW for scenario (TB-OO) is lower compared to other cases. In general the results indicate similar trends for the model predictions and real deployment runs, however, we observe some dips for TA/TB-OO as ONE is the lowest consistency level and does not provide a strict guarantee. Even though there are dips for certain values of L, the dips still have at least the high probability value of 90%.

5.2.3 Analysis Results for MR

Figure 10(a) shows the resulting probability of satisfying MR, where the probability is plotted against the issuing latency (L2) between W2 and R3. Again, it seems reasonable that all combinations of consistency levels offer high MR consistency, even with the lowest combination (O,O) that almost achieves a probability of 100%, and QUORUM reads gain more MR consistency than ONE reads. Surprisingly, it appears that the probability only relies on the consistency level of the reads, regardless of that of the concurrent write (W2): (O,O)/(O,Q) and (Q,O)/(Q,Q) with ONE/QUORUM reads have nearly the same probability trend. Note that we do not plot ALL reads because QUORUM reads already provide fairly stable probability of 100%.

Figure 10(c) shows the comparison of TB and TA regarding MR, where for simplicity we only list three lower combinations of consistency levels (with W2's consistency level increases, scenarios (TB/TA-QQ) will actually offer even higher consistency). Generally, all observations for TB apply to TA, and it seems fair to say that both strategies provide high MR consistency. TA's overall performance resembles TB's over issuing latency, but TA provides slightly higher consistency for some points.

Regarding the implementation-based evaluation (Figure 10(b) and (d)), we show the resulting probability of MR-consistent reads against L2 (Figure 5) for deployment runs regarding two strategies. Both the model predictions and deployment runs show high probability of satisfying MR. For the deployment runs, we observe that we get 100% MR consistency as the issuing latency

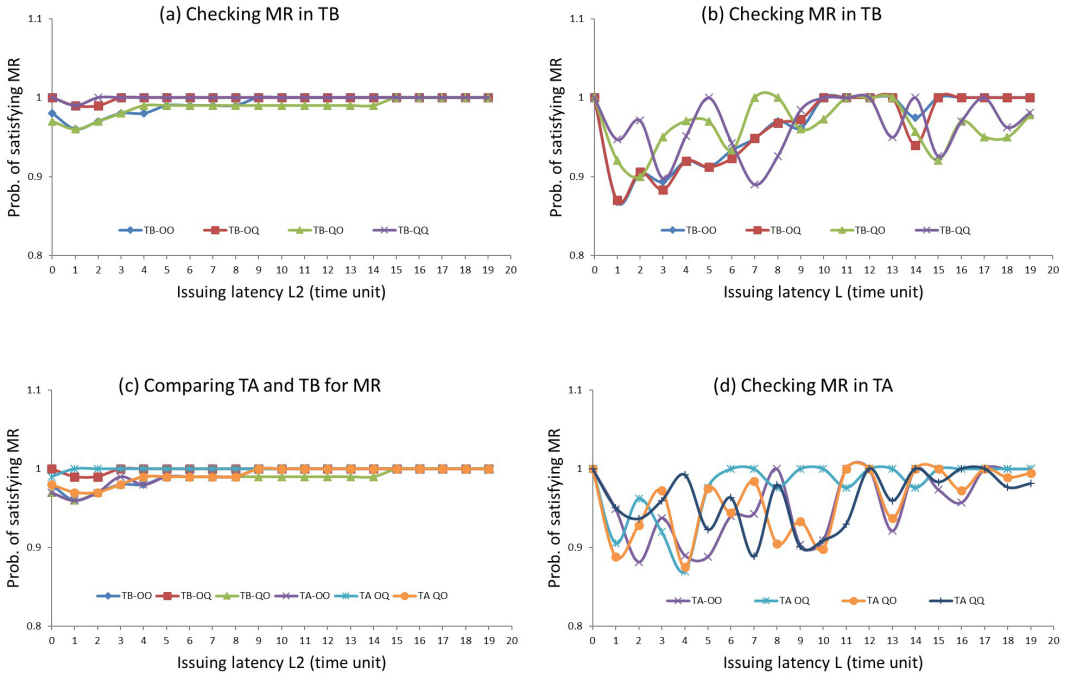


Figure 10 Probabilities of Satisfying MR by Statistical Model Checking (Left) and by Real Deployment Run (Right).

increases. For TA, the MR consistency fluctuates between 86% and 100%, but converges to 100% as the issuing latency increases. Note that the scenarios where the MR consistency is less than 100% are (OO) and (QO) for both strategies, and their MR consistency percentages are less compared to scenarios (QQ) and (OQ), which is expected.

5.2.4 Analysis Results for CP

Figure 11(a),(c),(e) show the resulting probability of satisfying CP by statistical model checking, where the probability (of R5 and R6 observing an ordered sequence of writes) is plotted against the issuing latency (`L`) between W1 and R5. By fixing the consistency level of writes, we can see how the consistency level of reads affect CP consistency over issuing latency: surprisingly, it appears that lower reads can achieve higher CP consistency. Specifically, `ONE` reads curve above `QUORUM` reads that curve above `ALL` reads in terms of each strategy. The reason is that reads with lower consistency level have lower latency (the interval between issuing time and finishing time for a read request), giving rise to higher chance to observe consecutive writes. Instead, higher latency reads allows more writes to reach the replicas during that interval, resulting in an inconsecutive observation (e.g., one anomaly occurs when R5 returns the value of W1 while R6 reads the value of W4).

We can compare the behaviors of TA and TB w.r.t. the choice of the three consistency levels: compared to TA, TB (1) conforms for `ONE` reads, (2) provides strictly higher CP consistency for `QUORUM` reads before some issuing latency, and, surprisingly, (3) gains more CP consistency only before some issuing latency for `ALL` reads. The reason is that CP aims at guaranteeing an ordered view of writes to the system instead of data freshness (e.g., R5 and R6 fetching W1 and W4 would be a CP-anomaly, though W4 is more recent than W2). Thus, after more and more writes have taken effect at the replicas (as `L` increases), TA with `ALL` reads is more likely to return the *convergent* value that respects the order.

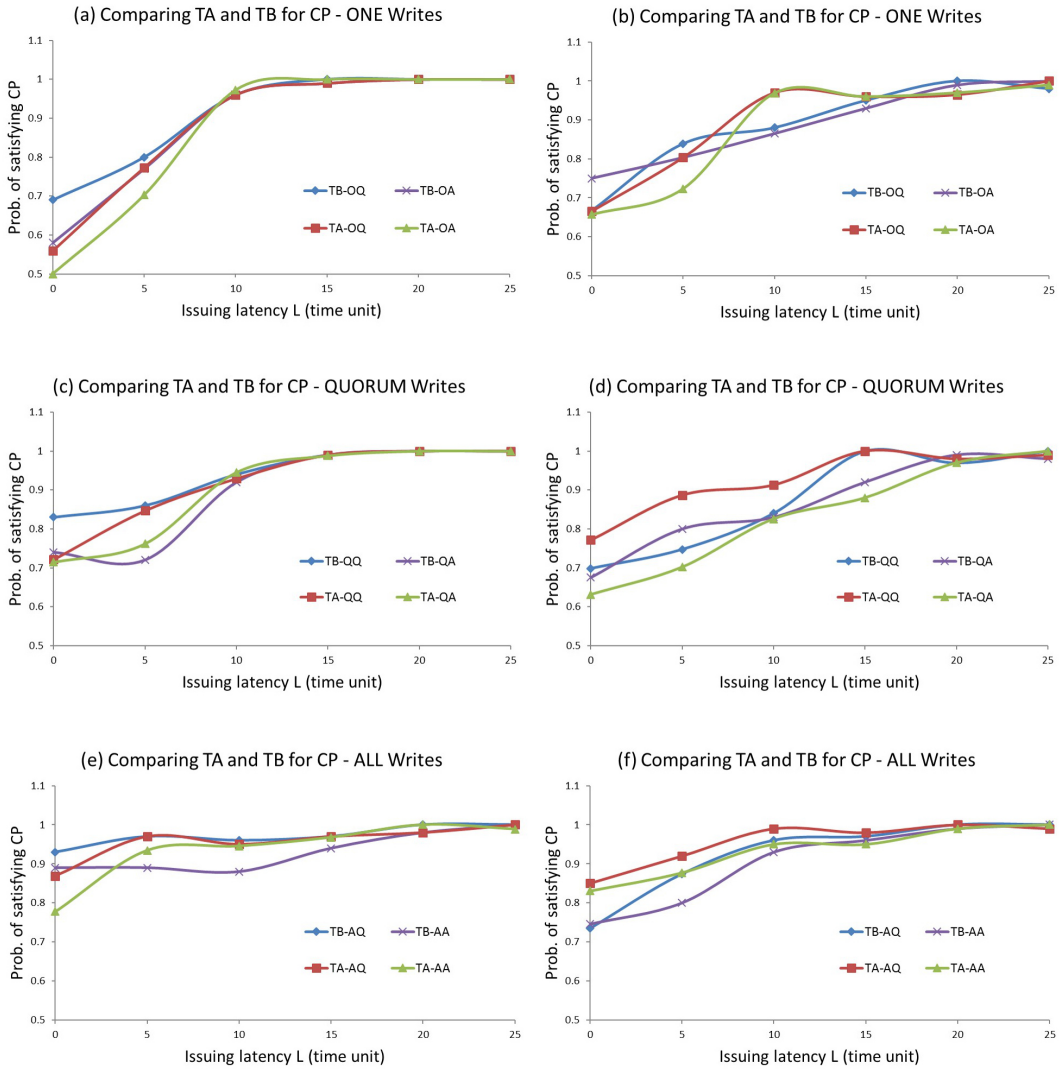


Figure 11 Probabilities of Satisfying CP by Statistical Model Checking (Left) and by Real Deployment Run (Right).

Note that all scenarios start with lower probability of satisfying CP (e.g., with `ONE` writes TB can only provide a probability down to 60%) because, when reads and writes are highly concurrent, reads would probably return the values of unordered writes.

Regarding the implementation-based evaluation (Figure 11(b),(d),(f)), we show the resulting probability of CP-consistent reads against `L` (Figure 6) for deployment runs regarding two strategies. The results indicate similar trends for the model predictions and real deployment runs. Both the model predictions and deployment runs show high probability of satisfying CP as the issuing latency `L` increases. For lower values of `L`, we observe CP consistency values between 60% and 80%, and, as `L` increases, the CP consistency value also increases to 70%–90%, gradually converging to 100%.

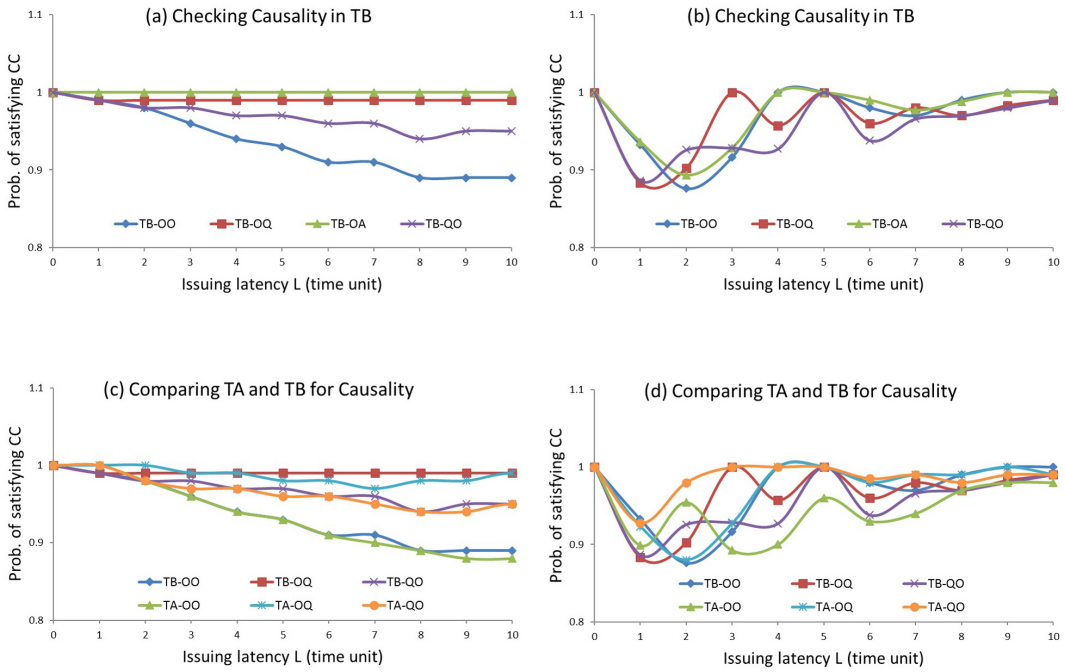


Figure 12 Probabilities of Satisfying CC by Statistical Model Checking (Left) and by Real Deployment Run (Right).

5.2.5 Analysis Results for CC

Figure 12(a) shows the resulting probability of satisfying CC, where the probability (of R5 returning the value of W4 provided R3 reads the value of W2) is plotted against the issuing latency (`L`) between W1 and R3. It is reasonable that all combinations of consistency levels achieve high CC consistency, even with the lowest combination `(O,O)` that provides a probability around 90%, and reads with higher consistency level gain more CC consistency (e.g., scenario (TB-OA) curves above scenario (TB-OQ) that curves above scenario (TB-QO/OO)). It is also straightforward to see that, with the consistency level of the preceding write (W4) from the same client increases, more consistency will be achieved (e.g., scenario (TB-QO) curves above scenario (TB-OO)). Note that all scenarios except (TB-OA) start with a probability of 100%, and gradually drop down and stabilize at a lower probability. The reason is that with a lower issuing latency there are fewer chances for R3 to read the value of W2 (if R3 does not return the value of W2, no causality will arise between these two operations) and therefore any value returned by R5 is considered to be CC-consistent; with an increasing issuing latency, R3 will be more likely to fetch the value of W2, which leaves the combination of consistency levels of W4 and R5 to be the only factor to affect the consistency (that also explains why the probability of satisfying CC stays fairly stable after some point).

Figure 12(c) shows the comparison of TB and TA in terms of CC, where for simplicity we only list three lower combinations of consistency levels due to the fact "`R5 + W4 > RF`". Both strategies achieve high CC consistency, and overlap for the combinations `(O,O)` and `(O,Q)`. Regarding `(O,Q)`, TA's performance resembles TB's over issuing latency, and both surpass each other slightly for some points.

Regarding the implementation-based evaluation (Figure 12(b) and (d)), we show the resulting probability of CC-consistent reads against `L` (Figure 7) for deployment runs regarding two strategies.

Both the model predictions and deployment runs show high probability of satisfying CC. For the model predictions, we observe that (TA-OO) and (TB-OO) have the lowest CC-consistency values around 90%, which is expected and also matches with our observations from the real deployment runs. In general, we observe that we get 90% - 100% CC-consistent reads, independent of the issuing latency.

5.2.6 Summary and Comparison

Our Cassandra model actually achieves much higher consistency (up to SC) than the promised EC, with `QUORUM` reads sufficient to provide up to 100% consistency in almost all scenarios. Comparing TA and TB, it seems fair to say that TA is not a competitive design alternative to TB in terms of the consistency models considered in this paper except CP, even though TA behaves close to TB in most cases; regarding CP, TA surpasses TB with `ALL` reads during a certain interval of issuing latency.

Our model, including the alternative design, is less than 1000 lines of code and the time to compute the probabilities for the consistency guarantees is 15 minutes (worst-case) on a 2.9 GHz Intel 4-Core i7-3520M CPU with 3.7 GB memory. The upper bound for model runtime depends on the confidence level of our statistical model checker (99% confidence level for all our experiments).

Both the model predictions and implementation-based evaluations reach the same conclusion: Cassandra provides much higher consistency than the promised EC, and TA does not improve consistency compared to TB in terms of the consistency models considered in this paper except for CP where TA provides higher consistency than TB with `ALL` reads during a certain interval of issuing latency. Note that the actual probability values from both sides might differ due to factors like hard-to-match experimental configurations, the inherent difference between statistical model checking and implementation-based evaluation[10], and processing delay at client/server side that our model does not include. However, the important observation is that the resulting *trends* from both sides are similar, leading to the same conclusions w.r.t. consistency measurements and strategy comparison.

Note that our experiments did not focus on the performance aspects of the TA and TB approaches in Cassandra. Our goal in this paper is to show that results derived from model-checking (using our model) agree reasonably well with experimental reality, e.g., see the comparisons and matches between our implementation results and the model results. The paper shows that this agreement holds true as far as various consistency models are concerned, and even if one changes how timestamps are used in responding to queries (comparing TA and TB).

6 Related Work

6.1 Model-based Performance Analysis of NoSQL stores

The work in [31] presents a queueing Petri net model of Cassandra parameterized by benchmarking only one server. The model is scaled to represent the characteristics of read workloads for different replication strategies and cluster sizes. Regarding performance, only response times and throughput are considered. The work in [20] benchmarks three NoSQL databases, namely Cassandra, MongoDB and HBase, by throughput and operation latency. Two simple high-level

[10] In general, implementation-based evaluation is based on a single trace of tens of thousands of operations (e.g., each YCSB client in our experiments performs 20,000 operations), while statistical model checking is based on sampling tens of thousands of Monte-Carlo simulations of several operations (that can be considered as a segment of the trace) up to a certain statistical confidence (e.g., our statistical model checking runs 40,000 Monte-Carlo simulations for the experimental scenario of CP which has only six operations).

queuing network models are presented that are able to capture those performance characteristics. Compared to both, our probabilistic model embodies the major components and features of Cassandra, and serves as the basis of statistical analysis of consistency with multiple clients and servers. Our model is also shown to be able to measure and predict new strategy designs by both statistical model checking and by checking the conformance of the model checking results with the code-based evaluation. Other recent work on model-based performance analysis includes [7], which applies multi-formalism modeling approach to the Apache Hive query language for NoSQL databases.

6.2 Experimental Consistency Benchmarking in NoSQL stores

The work in [32, 40, 10] proposes active and passive consistency benchmarking approaches for distributed NoSQL key-value stores, where operation logs are analyzed to find consistency violations. While these papers focus on consistency benchmarking using synthetic traces (e.g., generated using YCSB), the authors in [29] perform the first consistency benchmarking study of a large-scale production system (Facebook TAO system). YCSB+T [17] benchmarks isolation guarantees for NoSQL transactional databases, while the authors in [6] developed a benchmarking suite for interactive social networking applications running on top of NoSQL databases. The work in [5] proposes probabilistic notions of consistency to predict the data staleness, and uses Monte-Carlo simulations to explore the trade-off between latency and consistency in Dynamo-style partial quorum systems. Their focus is more on developing the theory of consistency models. However, we focus on building a probabilistic model for a key-value store like Cassandra itself, and our objective is to compare the consistency benchmarking results with the model-based predictions from our statistical model checking.

6.3 Rewriting Logic-based Analysis of Cloud Computing Systems

The work in [26] presents a formal model of the popular distributed key-value store Cassandra, and formally analyzes different consistency properties using Maude from a *qualitative* perspective. The work in [24] looks into the consistency issue for Cassandra only in terms of SC and RYW from a *quantitative*, statistical model checking perspective. This paper extends the previous results by providing quantitative analysis of three new consistency models: monotonic reads, consistent prefix and causal consistency. There is other recent work on rewriting logic-based analysis of *cloud computing* systems, including, e.g., [25], which formalizes RAMP transactions and their extensions and optimizations in rewriting logic and performs model checking verification of key properties using the Maude tool; [21, 22], which uses Maude and Real-Time Maude to define a formal model of Google's widely-replicated data store Megastore (a hybrid between a NoSQL store and a relational database) and to develop an extension of Megastore. These models were simulated for QoS estimation and model checked for functional correctness; [36], which formally models and analyzes availability properties of a ZooKeeper-based group key management service; [18], which proposes and analyzes DoS resilience mechanisms for cloud-based systems; [42], which gives formal semantics to the KLAIM language and uses it to specify and analyze cloud-based architectures;

7 Conclusion

Our main focus in this paper has been twofold: (i) to predict what consistency properties Cassandra can provide in actual practice by using statistical model checking; and (ii) to demonstrate the predictive power of our model-based approach in key-value store design by comparing statistical

model checking predictions with implementation-based evaluations. Our analysis is based on a formal probabilistic model of Cassandra. To the best of our knowledge, we are the first to develop such a formal model. The purpose of the alternative design was two-fold. First, we wanted to see if the alternative approach i.e., the Timestamp Agnostic (TA) approach would give better accuracy or not. Second, we also wanted to show that it was easier to try the alternative design with our formal model. This is one of the key benefits of building formal models so that we can quickly try alternative approaches and check their accuracy/performance. Our goal is not to do a systematic performance analysis of TA and TB approaches in Cassandra that would require larger setups not currently feasible for the statistical model checking analysis. Our goal instead is to show that results derived from model-checking (using our model) agree reasonably well with experimental reality, e.g., see the comparisons and matches between our implementation results and the model results. The paper shows that this agreement holds true as far as various consistency models are concerned, and even if one changes how timestamps are used in responding to queries (comparing TA and TB).

In this paper we have only investigated consistency guarantees in a quantitative manner. A natural next step would be to specify and quantify other performance metrics. Depending on the perspective (key-value store providers, users, or application developers), different metrics (e.g., throughput and operation latency) can be used to measure key-value store performance. We also plan to refine our model in order to quantify those metrics. While showing scalability is not the goal of this paper, we wish to do larger scale experiments in the future. There are various resource challenges related to scaling the model checking to larger system sizes (e.g., parallelizing it in the proper way) that we plan to address and solve this in our future work. More broadly, our long-term goal is to develop a library of formally specified executable components embodying the key functionalities of NoSQL key-value stores (not just Cassandra), as well as of distributed transaction systems [25]. We plan to use such components and the formal analysis of their performance to facilitate efficient exploration of the design space for such systems and their compositions with modest levels of effort and increased flexibility.

References

1 Gul A. Agha, José Meseguer, and Koushik Sen. Pmaude: Rewrite-based specification language for probabilistic object systems. *Electr. Notes Theor. Comput. Sci.*, 153(2):213–239, 2006. doi:10.1016/j.entcs.2005.10.040.

2 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995. doi:10.1007/BF01784241.

3 Musab AlTurki and José Meseguer. Pvesta: A parallel statistical model checking and quantitative analysis tool. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science – 4th International Conference, CALCO 2011, Winchester, UK, August 30 – September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 386–392. Springer, 2011. doi:10.1007/978-3-642-22944-2_28.

4 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. The potential dangers of causal consistency and an explicit solution. In Michael J. Carey and Steven Hand, editors, *ACM Symposium on Cloud Computing, SOCC'12, San Jose, CA, USA, October 14-17, 2012*, page 22. ACM, 2012. doi:10.1145/2391229.2391251.

5 Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *PVLDB*, 5(8):776–787, 2012. URL: http://vldb.org/pvldb/vol5/p776_peterbailis_vldb2012.pdf.

6 Sumita Barahmand and Shahram Ghandeharizadeh. BG: A benchmark to evaluate interactive social networking actions. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013. URL: http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper93.pdf.

7 Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Performance evaluation of nosql big-data applications using multi-formalism models. *Future Generation Comp. Syst.*, 37:345–353, 2014. doi:10.1016/j.future.2013.12.036.

8 Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: Facebook's photo storage. In Remzi H. Arpaci-Dusseau and Brad Chen, editors, *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 47–60. USENIX Association,

2010. URL: http://www.usenix.org/events/osdi10/tech/full_papers/Beaver.pdf.

9 Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, pages 267–280, 2010.

10 David Bermbach and Stefan Tai. Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In Karl M. Göschka, Schahram Dustdar, and Vladimir Tosic, editors, *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC 2011, Lisbon, Portugal, December 12-16, 2011*, page 1. ACM, 2011. doi:10.1145/2093185.2093186.

11 Eric A. Brewer. Towards robust distributed systems (abstract). In Gil Neiger, editor, *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA.*, page 7. ACM, 2000. doi:10.1145/343477.343502.

12 Cassandra, 2016. http://cassandra.apache.org.

13 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude – A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

14 Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In Joseph M. Hellerstein, Surajit Chaudhuri, and Mendel Rosenblum, editors, *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 143–154. ACM, 2010. doi:10.1145/1807128.1807152.

15 James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 261–264. USENIX Association, 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.

16 DB-Engines, 2016. http://db-engines.com/en/ranking.

17 Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Röhm. YCSB+T: benchmarking web-scale transactional databases. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 – April 4, 2014*, pages 223–230. IEEE Computer Society, 2014. doi:10.1109/ICDEW.2014.6818330.

18 Jonas Eckhardt, Tobias Mühlbauer, Musab AlTurki, José Meseguer, and Martin Wirsing. Stable availability under denial of service attacks through formal patterns. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering – 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, volume 7212 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2012. doi:10.1007/978-3-642-28872-2_6.

19 Jonas Eckhardt, Tobias Mühlbauer, José Meseguer, and Martin Wirsing. Statistical model checking for composite actor systems. In Narciso Martí-Oliet and Miguel Palomino, editors, *Recent Trends in Algebraic Development Techniques, 21st International Workshop, WADT 2012, Salamanca, Spain, June 7-10, 2012, Revised Selected Papers*, volume 7841 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2012. doi:10.1007/978-3-642-37635-1_9.

20 Andrea Gandini, Marco Gribaudo, William J. Knottenbelt, Rasha Osman, and Pietro Piazzolla. Performance evaluation of nosql databases. In András Horváth and Katinka Wolter, editors, *Computer Performance Engineering – 11th European Workshop, EPEW 2014, Florence, Italy, September 11-12, 2014. Proceedings*, volume 8721 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2014. doi:10.1007/978-3-319-10885-8_2.

21 Jon Grov and Peter Csaba Ölveczky. Formal modeling and analysis of google's megastore in real-time maude. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software – Essays Dedicated to Kokichi Futatsugi*, volume 8373 of *Lecture Notes in Computer Science*, pages 494–519. Springer, 2014. doi:10.1007/978-3-642-54624-2_25.

22 Jon Grov and Peter Csaba Ölveczky. Increasing consistency in multi-site data stores: Megastore-cgc and its formal analysis. In Dimitra Giannakopoulou and Gwen Salaün, editors, *Software Engineering and Formal Methods – 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings*, volume 8702 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2014. doi:10.1007/978-3-319-10431-7_12.

23 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

24 Si Liu, Son Nguyen, Jatin Ganhotra, Muntasir Raihan Rahman, Indranil Gupta, and José Meseguer. Quantitative analysis of consistency in nosql key-value stores. In Javier Campos and Boudewijn R. Haverkort, editors, *Quantitative Evaluation of Systems, 12th International Conference, QEST 2015, Madrid, Spain, September 1-3, 2015, Proceedings*, volume 9259 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2015. doi:10.1007/978-3-319-22264-6_15.

25 Si Liu, Peter Csaba Ölveczky, Muntasir Raihan Rahman, Jatin Ganhotra, Indranil Gupta, and José Meseguer. Formal modeling and analysis of RAMP transaction systems. In Sascha Ossowski,

editor, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 1700–1707. ACM, 2016. doi:10.1145/2851613.2851838.

26 Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal modeling and analysis of cassandra in maude. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering – 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2014. doi:10.1007/978-3-319-11737-9_22.

27 Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal modeling and analysis of Cassandra in Maude. Technical Report. Manuscript, 2014. URL: https://sites.google.com/site/siliunobi/icfem-cassandra.

28 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In Ted Wobber and Peter Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 401–416. ACM, 2011. doi:10.1145/2043556.2043593.

29 Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. Existential consistency: measuring and understanding consistency at facebook. In Ethan L. Miller and Steven Hand, editors, *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 295–310. ACM, 2015. doi:10.1145/2815400.2815426.

30 MongoDB, 2016. http://www.mongodb.org.

31 Rasha Osman and Pietro Piazzolla. Modelling replication in nosql datastores. In Gethin Norman and William H. Sanders, editors, *Quantitative Evaluation of Systems – 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings*, volume 8657 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2014. doi:10.1007/978-3-319-10696-0_16.

32 Muntasir Raihan Rahman, Wojciech M. Golab, Alvin AuYoung, Kimberly Keeton, and Jay J. Wylie. Toward a principled framework for benchmarking consistency. In Michael J. Freedman and Neeraj Suri, editors, *Proceedings of the Eighth Workshop on Hot Topics in System Dependability, HotDep 2012, Hollywood, CA, USA, October 7, 2012*. USENIX Association, 2012. URL: https://www.usenix.org/conference/hotdep12/workshop-program/presentation/rahman.

33 Redis, 2016. http://redis.io.

34 Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2005. doi:10.1007/11513988_26.

35 Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*, pages 251–252. IEEE Computer Society, 2005. doi:10.1109/QEST.2005.42.

36 Stephen Skeirik, Rakesh B. Bobba, and José Meseguer. Formal analysis of fault-tolerant group key management using zookeeper. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13-16, 2013*, pages 636–641. IEEE Computer Society, 2013. doi:10.1109/CCGrid.2013.98.

37 Doug Terry. Replicated data consistency explained through baseball. *Commun. ACM*, 56(12):82–89, 2013. doi:10.1145/2500500.

38 Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In Michael Kaminsky and Mike Dahlin, editors, *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP'13, Farmington, PA, USA, November 3-6, 2013*, pages 309–324. ACM, 2013. doi:10.1145/2517349.2522731.

39 Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009. doi:10.1145/1435417.1435432.

40 Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 134–143. www.cidrdb.org, 2011. URL: http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper15.pdf.

41 Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In David E. Culler and Peter Druschel, editors, *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*. USENIX Association, 2002. URL: http://www.usenix.org/events/osdi02/tech/white.html.

42 Martin Wirsing, Jonas Eckhardt, Tobias Mühlbauer, and José Meseguer. Design and analysis of cloud-based architectures with KLAIM and maude. In Francisco Durán, editor, *Rewriting Logic and Its Applications – 9th International Workshop, WRLA 2012, Held as a Satellite Event of ETAPS, Tallinn, Estonia, March 24-25, 2012, Revised Selected Papers*, volume 7571 of *Lecture Notes in Computer Science*, pages 54–82. Springer, 2012. doi:10.1007/978-3-642-34005-5_4.

43 Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006. doi:10.1016/j.ic.2006.05.002.

How Is Your Satellite Doing? Battery Kinetics with Recharging and Uncertainty*

Holger Hermanns[1], Jan Krčál[2], and Gilles Nies[3]

1 **Saarland University, Saarland Informatics Campus, Saarbrücken, Germany**
http://orcid.org/0000-0002-2766-9615
hermanns@cs.uni-saarland.de

2 **Saarland University, Saarland Informatics Campus, Saarbrücken, Germany**
http://orcid.org/0000-0002-3799-039X
krcal@cs.uni-saarland.de

3 **Saarland University, Saarland Informatics Campus, Saarbrücken, Germany**
http://orcid.org/0000-0002-2535-1590
nies@cs.uni-saarland.de

Abstract

The *kinetic battery model* is a popular model of the dynamic behaviour of a conventional battery, useful to predict or optimize the time until battery depletion. The model however lacks certain obvious aspects of batteries in-the-wild, especially with respect to the effects of random influences and the behaviour when charging up to capacity limits.

This paper considers the kinetic battery model with limited capacity in the context of piecewise constant yet random charging and discharging. We provide exact representations of the battery behaviour wherever possible, and otherwise develop safe approximations that bound the probability distribution of the battery state from above and below. The resulting model enables the time-dependent evaluation of the risk of battery depletion. This is demonstrated in an extensive dependability study of a nano satellite currently orbiting the earth.

2012 ACM Subject Classification Batteries, Stochastic processes, Reliability

Keywords and Phrases battery power, depletion risk, bounded charging and discharging, stochastic load, distribution bounds

Digital Object Identifier 10.4230/LITES-v004-i001-a004

Received 2016-01-09 **Accepted** 2016-12-05 **Published** 2016-12-23

Special Issue Editors Javier Campos, Martin Fränzle, and Boudewijn Haverkort

Special Issue Quantitative Evaluation of Systems

1 Introduction

A rechargeable battery is a physical object storing energy. Charging and discharging induces or is the result of chemical reactions inside the battery. Lithium chemistry is the technology of choice and has made rechargeable batteries become the backbone of our modern digital life. Yet, batteries are safety-critical. Wrong usage may imply injuries due to overheating, gas formation or spontaneous combustion. In addition, batteries are an obvious bottleneck for device operation, restricting performance and longevity of wireless operations, as well as journeys of electric vehicles. To understand and manage battery-run operations requires an adequate model of battery state

* This work is supported by the Transregional Collaborative Research Centre SFB/TR 14 AVACS, the 7th EU Framework Program under grant agreements 295261 (MEALS) and 318490 (SENSATION), by the Czech Science Foundation, grant no. P202/12/G061 and by the ERC Advanced Investigators Grant 695614 (POWVER).

reality		
electro-chemical model	(>50 parameters)	[13]
diffusion model	(3 parameters)	[31]
kinetic model	(3 parameters)	[28, 33]
Peukert model	(3 parameters)	[30]
linear model	(1 parameter)	
observables	(voltage, temperature)	

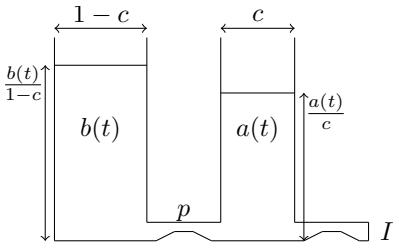


Figure 1 Battery model overview (left) and visualisation of the kinetic battery model (right).

and battery behaviour. Different modelling approaches for predicting battery performance have been proposed [32, 6] based on a model landscape summarised on the left of Figure 1, where each model is known to be an abstraction of its upper neighbour [25, 23].

Detailed explanations of the various models will be given in Section 2. Intuitively, the linear model corresponds to a simple well holding liquid, the charge. This is is the view typically displayed to smart phones users, in the form of a percentage value. The diffusion model treats the distribution of electrical charge between cathode and anode as a continuum, while its first-order approximation, the *kinetic battery model*, KiBaM, separates the charge in two parts, available charge and bound charge. The latter can be visualised as two wells interconnected by a pipe, as depicted on the right of Figure 1, where only the available charge may be consumed instantaneously, while bound charge is converted into available charge as time passes. Given a constant load, the KiBaM represents the battery *state-of-charge* (SoC) by two coupled differential equations, one for each well. Unlike the linear model (and the Peukert model), the KiBaM can capture two important real phenomena; the *rate capacity effect* and the *recovery effect*. The former effect describes the fact that if continuously discharged, a high discharge rate will cause the battery to provide less energy before depletion than a lower discharge rate. Thus a battery's effective capacity depends on the rate at which it is discharged. The latter effect stands for the battery recovers to some extent during periods of no or little discharge. Both effects are decisive operational phenomena known across electro-chemical batteries, rooted in their physical layout where the chemical reactions related to charging and discharging span between cathode and anode, and are dis-equilibrating the chemical substrate balance. Indeed, empirical evaluations show that this model provides a good approximation of the battery SoC across various battery types [25, 23].

Our contribution. The original KiBaM does not take capacity limits into consideration, it can thus be interpreted as assuming infinite capacity. Reality is unfortunately different. When studying the KiBaM operating with capacity limits, it becomes apparent that charging and discharging are *not* dual to each other, simply because a full battery keeps operating, in contrast to an empty one. However, opposite to the discharging process, the charging process near capacity limits has not received dedicated attention in the literature. That problem is attacked in the present paper.

Although directly expressible as a function of time, the behaviour at capacity limits cannot be computed exactly. For this scenario we therefore resort to under- and over-approximating state of charge (SoC) evolutions, that serve as upper and lower bounds of the exact SoC evolution.

Furthermore, statistical results obtained by experimenting with real of-the-shelf batteries suggest considerable variances in actual performance [7], likely rooted in manufacturing and wear differences. This observation asks for a stochastic re-interpretation of the classical KiBaM to take the statistically observed SoC spread into account on the model level, and this is what the present paper develops – in a setting with capacity limits, charging and discharging. It views the KiBaM as a transformer of the continuous probability distribution describing the SoC at any real time

point, thereby also supporting uncertainty and noise in the load process.

The stochastic re-interpretation and the extension by capacity limits in combination, allow us to derive SoC distributions that bound the actual distribution of the SoC from above and below in a safe way. These bounding distributions can be used to determine an interval enclosing the cumulative risk of battery depletion for any given time point.

We apply this approach to an in-depth case study of the Danish nano satellite GomX–1 currently orbiting the earth in low orbit [20]. From the satellite's hardware specification and extensive in-flight telemetry logs provided by its manufacturer GomSpace, a probabilistic workload model is derived and superposed with a periodic deterministic charging load, representing the infeed from on-board solar panels. Our technique then enables us to perform an effective quantitative analysis of the satellite's power budget, with a particular focus on the battery depletion risk over large mission times. The interplay of the resulting battery model and the imposed load can be viewed as a particular stochastic hybrid system [1, 3, 4, 8, 12, 37], developed here without discretising time. We have found that general purpose tools for this problem domain [36, 17, 43] are at present not capable to provide such answers, as we will explain.

The genuine contributions of the paper are:

(i) The interpretation of the KiBaM as a transformer of SoC distributions,
(ii) developed without discretising time,
(iii) considering both charging and discharging in the context of capacity limits,
(iv) using under- and over-approximations where needed to get correctness guarantees,
(v) applied in the power budget analysis of a low-earth orbiting nano satellite.

Related work. Haverkort and Jongerden [23] review broad research on various battery models of different natures, ranging from electro-chemical models, electrical circuit models, stochastic models to analytical models. The conclusion is very plain: The most accurate models are the electro-chemical ones, although their usage requires expert knowledge about batteries. For integration with a workload model to carry out performance analysis, analytical models are best suited as they allow for analytical expressions of the battery lifetimes under a load process, while still capturing the most important non-linear effects of real batteries.

They particularly discuss *stochastic* battery models [33, 10] which view the KiBaM for a given load as a stochastic process, unlike our (more accurate) view as a deterministic transformer of the randomized initial conditions of the battery. Furthermore, in this survey, the problem of charging up to capacity limits does not get dedicated attention.

Battery capacity has been addressed only by Boker *et al.* [5]. They considered a discretized, unbounded KiBaM together with a possibly non-deterministic and cyclic load process, synthesizing initial capacity requirements to power the process safely. Hence, capacity is here understood as an over-dimensioned initial condition and not as a truly limiting charging bound.

Random loads on a battery, generated by a continuous-time Markov chain, have been previously studied by Cloth *et al.* [10]. Their setting cannot be easily extended by charging since they view the available and bound charge levels as two types of *accumulated reward* in a reward-inhomogeneous continuous time Markov chain.

An extension of the KiBaM to *scheduling* has been considered by Jongerden *et al.* [24]. They compute optimal schedules for multiple batteries in a discretized setting with only discharging. This has been taken up and improved using techniques from the planning domain [14].

Organisation of the paper. Section 2 introduces the original (deterministic, unlimited) kinetic battery model. In Section 3 we view this unlimited KiBaM as a transformer of probability density functions, resulting in a stochastic, unlimited model. Section 4 considers lower and upper capacity

limits for the deterministic model. This development is lifted to the stochastic interpretation in Section 5, arriving at a stochastic and limited model. Section 6 introduces a probabilistic workload model together with algorithms to compute the SoC of a stochastic limited KiBaM under such a workload. In Section 7 we present a discretisation algorithm that allows for efficient and tight approximations of a SoC distribution after being subject to a workload for a certain amount of time. Finally, in Section 8, we demonstrate the efficiency and relevance of our findings by analysing the power budget of a Danish nano satellite currently orbiting the planet.

This paper is a substantially enhanced and extended version of paper [22]. Apart from a more extensive exposition, the enhancements comprise all developments and results concerning over-approximations, a connection to synchronous data flow models, as well as a thorough and detailed description of the nano satellite case.

2 The Kinetic Battery Model

As discussed in Section 1, batteries in-the-wild exhibit two non-linear effects widely considered to be the most important ones to capture: the *rate capacity effect* and the *recovery effect*. In the sequel we introduce the *kinetic battery model* KiBaM as the simplest model capturing these effects, and place it in the context of other candidates to model a battery, summarized in Figure 1.

The linear model. Also called the *ideal battery*, this model views a battery as one well of capacity cap that is decreased proportionally to a load I that is imposed on the battery. Thus, the lifetime of a full battery under load I can naturally be expressed by cap/I. While easy to handle, the linear battery model neither captures the recovery of batteries nor the rate capacity effect.

Peukert model. An extension of the ideal battery is provided by *Peukert's law*. In this, parameters a and b characterise the lifetime of a full battery under load I as a/I^b. For $a = \mathsf{cap}$ and $b = 1$, this corresponds to an ideal battery, although parameters fitted through experiments generally result in a being a bit smaller than cap and b being slightly larger than 1. Peukert's law captures the rate capacity effect, but neglects the recovery effect.

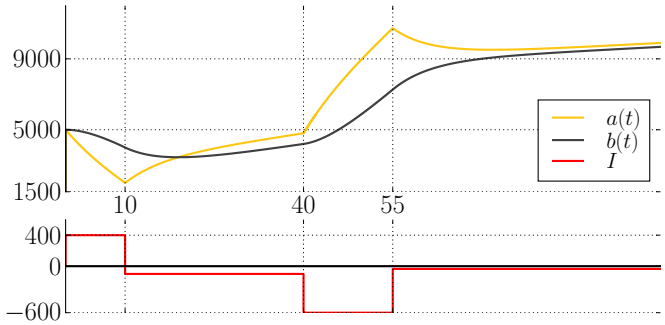
The electrochemical-model. Together with its accompanying simulation tool DUALFOIL [29], the highly parametrisable electro-chemical battery model is, in its own right, widely considered as the reference "reality" to check the faithfulness and accuracy of other models.

The diffusion model. The *diffusion model* of Rakhmatov and Vrudhula [25] describes the ion concentration along the width of a battery as a continuous quantity. A full battery exhibits equal concentration along the battery, while a discharge causes a decrease of the concentration near the discharging electrode. This, in turn, causes a gradient that makes the ions diffuse towards the electrode. Thus during periods of rest the ion concentration tends to equalise over the width of a battery, inducing a recovery. During periods of high discharge, the diffusion cannot keep up causing premature depletion; the rate capacity effect. The model allows for analytical expressions for the battery lifetime as well and exhibits a very high degree of precision against the electro-chemical model.

The kinetic battery model. The *kinetic battery model* (KiBaM) can be viewed as a discretised diffusion model by dividing the stored charge into two parts, the *available* charge and the *bound* charge and can actually be proven to be a first-order approximation of the diffusion model. When the battery is strained only the available charge is consumed instantly, while the bound charge is slowly converted to available charge by diffusion. This diffusion between available and bound charge can take place in either direction depending on the amount of both types of energy stored in the battery. Both non-linear effects are captured for the exact same reason as for the diffusion model: the relatively slow conversion of bound charge into available

charge or vice versa. Due to its simplicity and accuracy relative to the more complex diffusion model [23] and therefore also relative to the DUALFOIL electro-chemical model simulator, we focus on the kinetic battery model in this paper.

▶ **Example 1.** We illustrate the evolution of the state of charge of the KiBaM as time passes under the assumption of symmetry of charging and discharging below.



The initially available charge decreases heavily due to the load 400 but the restricted diffusion makes the bound charge decrease only slowly up to time 10; after that the battery undergoes a mild recharge, and so on. At all times the bound charge approaches the available charge by a speed proportional to the difference of the two values.

Coupled differential equations. The KiBaM can be visualized as two wells holding liquid, interconnected by a pipe that represents the diffusion of the two types of charge, as depicted on the right of Figure 1. The available charge well is exposed directly to the load I and connected to the bound charge well by a pipe of width p. Formally, the KiBaM is characterized by two coupled differential equations

$$\dot{a}(t) = -I + p\left(\frac{b(t)}{1-c} - \frac{a(t)}{c}\right), \qquad \dot{b}(t) = p\left(\frac{a(t)}{c} - \frac{b(t)}{1-c}\right). \tag{1}$$

Here, the functions $a(t)$ and $b(t)$ describe the available and bound charge at time t respectively, $\dot{a}(t)$ and $\dot{b}(t)$ their time derivatives, I is a *load* on the battery. We refer to the parameter p as the *diffusion rate* between both wells, while parameter $c \in [0,1]$ corresponds to the width of the available charge well, and $1-c$ is the width of the bound charge well. Intuitively, $a(t)/c$ and $b(t)/(1-c)$ are the level of the fluid stored in the available charge well and the bound charge well, respectively. In the following, we take a closer look at the properties of this concrete form of a dynamical system. In the end, this allows us to obtain tailor-made efficient analysis algorithms.

It is possible to derive a solution of the ODEs at time t when applying load I, for instance by using Laplace transforms. We can express it as a vector valued linear mapping $\mathbf{K}_{t,I}$ taking the initial available and bound charge a_0 and b_0 as argument:

$$\mathbf{K}_{t,I}\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} \mathrm{q}_a & \mathrm{r}_a & \mathrm{s}_a \\ \mathrm{q}_b & \mathrm{r}_b & \mathrm{s}_b \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ b_0 \\ I \end{bmatrix} \qquad \text{where} \qquad \begin{aligned} \mathrm{q}_a &= (1-c)\mathrm{e}^{-kt} + c, \\ \mathrm{r}_a &= -c\,\mathrm{e}^{-kt} + c, \\ \mathrm{s}_a &= \frac{(1-c)(\mathrm{e}^{-kt}-1)}{k} - t \cdot c \end{aligned}$$

and $\mathrm{q}_b = 1 - \mathrm{q}_a$, $\mathrm{r}_b = 1 - \mathrm{r}_a$, $\mathrm{s}_b = -t - \mathrm{s}_a$ and finally $k = p/c(1-c)$. The coefficients s_a and s_b of I do not sum to 1, because the non-zero load I makes the total power in the battery change. The above definition of $\mathbf{K}_{t,I}$ is a vector valued reformulation of equations found in [28].

As all vectors appearing in this paper are column vectors, we also denote them by semicolon notation $[a;b]$. Furthermore, whenever we compare two vectors, e.g., $[a;b] \leq [a';b']$, we interpret

the order component-wise. When $[a_0; b_0]$ and I are clear from context, we denote the SoC $\mathbf{K}_{t,I}[a_0; b_0]$ at time t also simply by $[a_t; b_t]$.

▶ **Example 2.** We can use the function $\mathbf{K}$ to obtain the final SoC for our example (for $k = 1/100$, $c = 1/2$, and $\circ$ denoting function composition) by

$$\mathbf{K}_{45,-35} \circ \mathbf{K}_{15,-600} \circ \mathbf{K}_{30,-100} \circ \mathbf{K}_{10,400}[5000; 5000] \approx \mathbf{K}_{45,-35}[10732; 7268],$$

and with the last step in more details (denoting $\mathrm{e}^{-\frac{44}{100}}$ by E),

$$= \begin{bmatrix} \frac{1}{2}E + \frac{1}{2} & -\frac{1}{2}E + \frac{1}{2} & 50E - 50 - \frac{44}{2} \\ -\frac{1}{2}E + \frac{1}{2} & \frac{1}{2}E + \frac{1}{2} & 50 - 50E - \frac{44}{2} \end{bmatrix} \cdot \begin{bmatrix} 10732 \\ 7268 \\ -35 \end{bmatrix} = \begin{bmatrix} -18E + 6480 \\ 18E + 8020 \end{bmatrix} \approx \begin{bmatrix} 9881 \\ 9659 \end{bmatrix}.$$

The first summands on the last line (with E) stand for the spread of the values before the recovery effect converges (as $E \to 0$ for $t \to \infty$). The second summands are different due to non-zero load I causing $b_t - a_t$ to converge to I/k (for $c = 1/2$).

Powering a task. A standard problem in battery modelling and evaluation is to find out whether a task can be performed with a given *positive* state of charge without depleting the battery, where positive is to be understood componentwise. A task is a pair (T, I) with T being the task execution time, and I representing the load, imposed for duration T.

▶ **Definition 3** (**K**-powering a task)**.** For an execution time T and a load I, we say that a battery with a positive SoC $[a_0; b_0]$ $\mathbf{K}$*-powers a task* (T, I) iff $\forall\, 0 < t \leq T : a_t > 0$.

Let us stress that the SoC of the battery evolves in negative numbers in the same way as in positive numbers because the differential equations do not have any explicit bounds. Furthermore, it is not monotonic with respect to time in the conventional sense.

▶ **Example 4.** In our example, the bound charge is not monotonic on the interval $[10, 40]$, the available charge is not monotonic on $[55, 100]$. However, for instance, on $[40, 55]$, available charge is the first to get above the value 9000 (and never crosses the boundary back down again).

We observe that the KiBaM is monotonic with respect to *crossing a boundary* κ when both wells start beyond this boundary.

▶ **Lemma 5.** *For any* $I \in \mathbb{R}$, $\kappa \in \mathbb{R}$, $\rhd \in \{<, >\}$, $[a_0; b_0] \rhd [c\kappa; (1-c)\kappa]$ *and for* $t \in \mathbb{R}_{>0}$ *such that* $a_t = c\kappa$ *we have*

- $b_t \rhd (1-c)\kappa$ *(available charge is always the first to cross a boundary);*
- $a_T \not\rhd c\kappa$ *for all* $T > t$ *(available charge never crosses back for a given load).*

Intuitively speaking, the first property states that the available charge is always the first to cross a boundary, the second property states that when the available charge crosses a boundary it never returns back (for a given load).

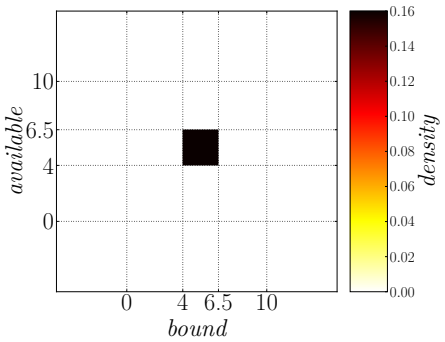
As a direct consequence of Lemma 5, we can easily find out whether the battery $\mathbf{K}$-powers a task (T, I) by just observing the SoC at time point T.

▶ **Lemma 6.** *A battery with a positive SoC* $[a_0; b_0]$ $\mathbf{K}$*-powers a task* (T, I) *iff* $[a_T; b_T] > [0; 0]$.

3 Random KiBaM

In order to consider the KiBaM as a stochastic object, it appears natural to consider the vector $[a_0; b_0; I]$ as being random. This reflects the perturbations of the load and of the initial SoC of the batteries. The latter is a real phenomenon, rooted in wear and manufacturing variances [9]. We thus assume the initial SoC to be random variables A_0, B_0 jointly distributed according to a density function f_0, while the load on the battery is a random variable I independent of the SoC, endowed with a probability density function g.

▶ **Example 7.** Instead of a single (Dirac) SoC, we now consider that the joint density f_0 of the charge is, say, uniform over the area $[4, 6.5] \times [4, 6.5]$ as depicted below.



Here the values of the two-dimensional density are expressed using colours. Using similar plots, we shall illustrate how the SoC distribution evolves as the time passes on this particular example.

Evolution over time. We are interested in the random vector expressing the SoC after some time T for a *constant* (but random) load I. This is given by

$$[A_T; B_T] \coloneqq \mathbf{K}_{T,I}[A_0; B_0]. \tag{2}$$

The core tool for studying the joint density of $[A_T; B_T]$ is the transformation law for random variables, which enables the construction of unknown density functions from known ones if given the relation between the corresponding random variables. Formally, for every d-dimensional random vector $\mathbf{X}$ and every injective and continuously differentiable function $g : \mathbb{R}^d \to \mathbb{R}^d$, we can express the density function of $\mathbf{Y} \coloneqq g(\mathbf{X})$ at value y in the range of g as

$$f_{\mathbf{Y}}(y) = f_{\mathbf{X}}\left(g^{-1}(y)\right) \cdot \left|\det\left(J_{g^{-1}}(y)\right)\right| \tag{3}$$

where $J_{g^{-1}}(y)$ denotes the *Jacobian* of g^{-1} evaluated at y. However, the mapping (2) is not invertible, thus we cannot directly apply the transformation law. Instead, we express the joint density conditioned by the random load I attaining some arbitrary but fixed value i. For this fixed i, we can exploit the specific structure of the KiBaM to express the transformation using an invertible linear mapping

$$\mathbf{K}_{T,i}\begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \begin{bmatrix} \mathrm{q}_a & \mathrm{r}_a \\ \mathrm{q}_b & \mathrm{r}_b \end{bmatrix} \cdot \begin{bmatrix} A_0 \\ B_0 \end{bmatrix} + \begin{bmatrix} \mathrm{s}_a \\ \mathrm{s}_b \end{bmatrix} \cdot i.$$

A straightforward inversion of the mapping results in

$$\mathbf{K}^{-1}_{T,i}\begin{bmatrix} a \\ b \end{bmatrix} = \mathrm{e}^{kT}\begin{bmatrix} \mathrm{r}_b & -\mathrm{r}_a & \mathrm{r}_a\mathrm{s}_b - \mathrm{r}_b\mathrm{s}_a \\ -\mathrm{q}_b & \mathrm{q}_a & \mathrm{q}_b\mathrm{s}_a - \mathrm{q}_a\mathrm{s}_b \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ i \end{bmatrix}.$$

Applying (3) we arrive at the joint density of $[A_T; B_T]$ conditioned by $I = i$

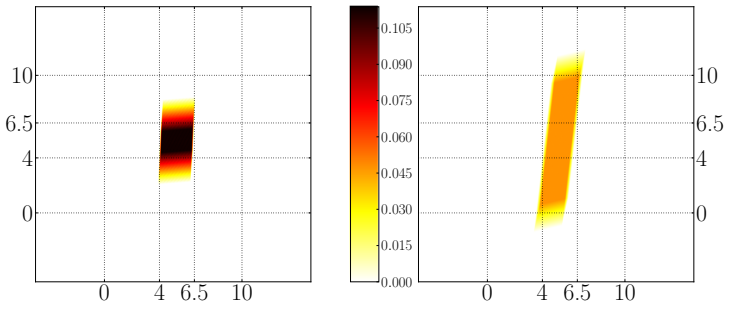
$$f_T(a, b \mid i) = f_0\Big(\mathbf{K}_{T,i}^{-1}[a; b]\Big) \cdot \left|\mathrm{e}^{kT}\right|$$

where e^{kT} is the determinant of the Jacobian of $\mathbf{K}_{T,i}^{-1}$. Interestingly, it is constant in a, b and i, it only depends on T. It is also non-negative for $T \geq 0$ as $k > 0$. Finally we get rid of the conditional $I = i$ by marginalizing the variable $[A_T; B_T]$. Intuitively, this averages the conditional densities over the distribution g of I, obtaining thus the following.

▶ **Lemma 8.** *Let T be execution time and g be load density. For an initial SoC f_0 over $[A_0; B_0]$ and task (T, g), the joint distribution of $[A_T; B_T]$ is absolutely continuous with density f_T given by*

$$f_T(a, b) = \mathrm{e}^{kT} \int_{\mathbb{R}} f_0\Big(\mathbf{K}_{T,i}^{-1}[a; b]\Big) \cdot g(i)\,\mathrm{d}i.$$

▶ **Example 9.** We return to our example assuming the density g of the load being uniform between $[-0.1, 0.1]$. We can compute the SoC of the battery after task $(20, g)$, displayed on the left, and $(60, g)$, displayed on the right. Here, we arbitrarily chose the parameters $c = 0.5$ and $p = 0.002$.



Probability of powering a task. We are now in the position to transfer the problem of powering a task to the stochastic setting. We say that a density f_0 is *positive* if it supports only positive SoCs, i.e. for any a, b such that either $a \leq 0$ or $b \leq 0$ we have $f_0(a, b) = 0$.

▶ **Definition 10** (Probability of $\mathbf{K}$-powering a task)**.** For an execution time $T > 0$ and a load density g, we say that the battery (with positive initial SoC f_0) $\mathbf{K}$-*powers* a task (T, g) *with probability (at least)* $p > 0$ if

$$\mathbf{Pr}[\forall\, 0 \leq t \leq T : A_t > 0] \geq p.$$

Due to the monotonicity of KiBaM in the sense of Lemma 5, this is equivalent to observing the probability of depletion *only* at time T. From Lemma 8 we obtain the following.

▶ **Lemma 11.** *A battery with SoC f_0 $\mathbf{K}$-powers with probability $p > 0$ a task (T, g) if and only if*

$$\iint_{\mathbb{R}^2_{>0}} f_T(a, b)\,\mathrm{d}a\,\mathrm{d}b \;\geq\; p.$$

▶ **Example 12.** Thanks to the lemma, it suffices to perform the integration on the densities displayed in the previous plots in this running example. The probability to power the tasks $(20, g)$ is 1, for the task $(60, g)$ it is just ≈ 0.968.

4 Deterministic Limited KiBaM With Recharging

Both charging and discharging are well supported by the theory developed so far, as charging has occurred in our examples in the form of negative loads. What is not treated in the theory yet is a capacity limit. This however is an obvious real constraint in applications employing rechargeable batteries. To the best of our knowledge, charging in KiBaM while respecting its capacity restrictions has not been addressed even in the deterministic case. Thus, we dedicate this section to developing the deterministic setting first. In the next section, we extend the theory to the randomized setting.

We assume that the battery has capacity cap divided into capacity $a_{\mathsf{max}} = c \cdot \mathsf{cap}$ of the available charge well and capacity $b_{\mathsf{max}} = (1-c) \cdot \mathsf{cap}$ of the bound charge well. Charging and discharging are not fully symmetric: A battery with empty available charge can no longer power its task, contrary to a battery with full available charge that *continues to operate*. We thus need to consider its further charging behaviour.

When the available charge is at its capacity $a_{\mathsf{max}} = c \cdot \mathsf{cap}$ and is still further charged by a *sufficiently* high charging current, its value stays constant and only the bound charge increases due to diffusion. Hence, for any $t \geq 0$ we have $a(t) = c \cdot \mathsf{cap}$ and thus $\dot{a}(t) = 0$. The equation for the bound charge from (1) is modified to an ODE

$$\dot{b}(t) = p\left(\mathsf{cap} - \frac{b(t)}{1-c}\right). \tag{$\bar{1}$}$$

Staying at the upper limit. The differential equation above describes the behaviour of the battery at time t only if the incoming current to available charge well is *sufficient* to compensate the diffusion, i.e. $-I \geq \dot{b}(t)$. Since I is constant and the diffusion is decreasing over time, the charging current is sufficient at all times if and only if it is sufficient at time 0, i.e. $-I \geq \dot{b}(0)$.

For an initial bound charge b_0 we define the condition whether the charging current is sufficient by

$$I \leq \bar{I}(b_0) \coloneqq p\left(\frac{b_0}{1-c} - \mathsf{cap}\right) \tag{4}$$

which requires the initial bound charge to be close enough to its capacity so that the charging current overcomes the diffusion.

By solving the ODE ($\bar{1}$) and using Inequation (4), we obtain the following result.

▶ **Lemma 13.** *Let $T > 0$ and b_0 such that $I \leq \bar{I}(b_0)$. A battery with a SoC $[a_{\mathsf{max}}; b_0]$ reaches after the task (T, I) the state of charge $\left[a_{\mathsf{max}}; \bar{b}_T(b_0)\right]$ where*

$$\bar{b}_T(b_0) = \mathrm{e}^{-ckT} b_0 + \left(1 - \mathrm{e}^{-ckT}\right) \cdot b_{\mathsf{max}} \tag{5}$$

and k again stands for $p/\left(c \cdot (1-c)\right)$.

We notice that the resulting bound charge evolution $\bar{b}_T(b_0)$ does not further depend on I, i.e. one cannot make the battery charge faster by increasing the charging current. Furthermore, for a fixed b_0, the curve of $t \mapsto \bar{b}_t(b_0)$ is a negative exponential starting from the point b_0 with the full capacity b_{max} of the bound charge being its limit. Thus, Lemma 13 also reveals that the bound charge in finite time never reaches its capacity and there is no need to describe this situation separately. Finally, we denote analogously by $\bar{\mathbf{K}}_T[a_0; b_0] = \left[a_0; \bar{b}_T(b_0)\right]$ the linear mapping describing the behaviour at the upper limit.

Hitting the upper limit. When charging with a given constant load I, we have two modes of behaviour of the battery:

(i) *before* the available charge hits a_{max} and

(ii) *after* it hits (and stays at) a_{max}.

The remaining question is *when* it hits that capacity limit. For a given initial state $[a_0; b_0] < [a_{\mathsf{max}}; b_{\mathsf{max}}]$ and a load I, this amounts to finding $\bar{t} \in \mathbb{R}_{>0}$ such that $a_{\bar{t}} = a_{\mathsf{max}}$. This induces that the following equation can be derived from $\mathbf{K}_{\bar{t},I}[a_0; b_0]$,

$$u \cdot \mathrm{e}^{-k\bar{t}} + v \cdot \bar{t} + w = a_{\mathsf{max}}$$

where $u = a_0\,(1-c) - b_0 c + (c+1) \cdot I/k$, $v = -Ic$, and $w = a_{\mathsf{max}} - a_0 c - b_0 c - (1-c) \cdot I/k$. In this equation, $\bar{t}$ appears in an exponential as well as in a linear term. This is characteristic for a non-elementary function $\mathcal{W}$ called *product logarithm* which can express the solution as

$$\bar{t} = -\mathcal{W}\Big(\frac{u}{v} \cdot \mathrm{e}^{-\frac{w}{v}}\Big) - \frac{w}{v}. \tag{6}$$

The product log function can approximated by numerical methods [11].

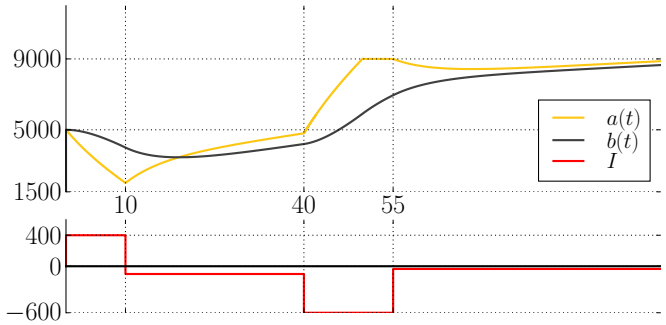
Integrating the two modes of behaviour. All the previous building blocks allow us to express easily the SoC of a deterministic KiBaM after powering a given task (T, I) when considering capacity limits. We define it as

$$\mathbf{K}^{\square}_{T,I}[a_0; b_0] \coloneqq \begin{cases} \mathbf{K}_{T,I}[a_0; b_0] & \text{if } a_0 > 0 \wedge 0 < a_T \leq a_{\mathsf{max}}, \\ \bar{\mathbf{K}}_t \circ \mathbf{K}_{\bar{t},I}[a_0; b_0] & \text{if } a_0 > 0 \wedge \quad a_T > a_{\mathsf{max}}, \\ [0; 0] & \text{if } a_0 = 0 \vee 0 \geq a_T \end{cases}$$

where $\bar{t}$ is the largest solution of (6) and $t = T - \bar{t}$.

The first two cases in $\mathbf{K}^{\square}_{T,I}$ match the behaviour explained earlier thanks to Lemma 5. Whenever the upper limit is hit, it will never be crossed back with the given I and thus also I is sufficient according to (4).

▶ **Example 14.** If we put a limit of 9000 to the previous scenario, the battery ends up with a slightly smaller charge at time 100. The computation of the final SoC changes only in the interval $[40, 55]$. Here, instead of $\mathbf{K}_{15,-600}$, we apply $\mathbf{K}_{\bar{t},-600}$ for the first $\bar{t} \approx 7.8$ time units, followed by $\bar{\mathbf{K}}_{15-\bar{t}}$.



Similarly to Section 2, we establish the notion of $\mathbf{K}^{\square}$-powering a task.

▶ **Definition 15** ($\mathbf{K}^{\square}$-Powering a task)**.** A battery with a positive SoC $[a_0; b_0]$ $\mathbf{K}^{\square}$-*powers* a task (T, I) if $\forall\, 0 < t \leq T : a_t > 0$ where each $[a_t; b_t]$ denotes $\mathbf{K}^{\square}_{t,I}[a_0; b_0]$.

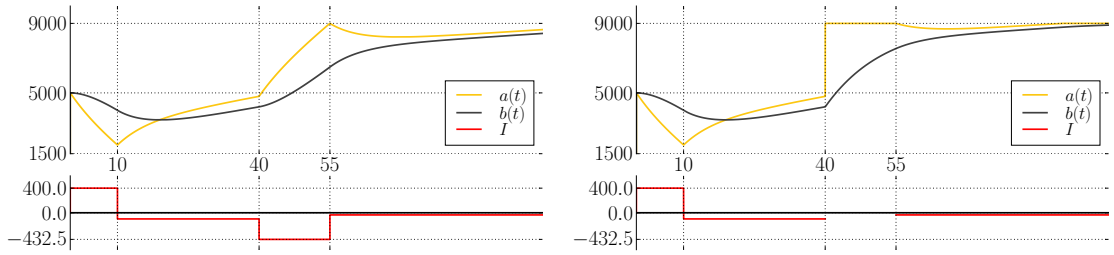
▶ **Lemma 16.** *A battery with a positive SoC* $[a_0; b_0]$ $\mathbf{K}^{\square}$*-powers a task* (T, I) *if and only if* $\mathbf{K}^{\square}_{T,I}[a_0; b_0] > [0; 0]$.

Computing $\mathbf{K}^{\square}$ efficiently. The problematic part in computing $\mathbf{K}^{\square}$ is the case $a_t > a_{\mathsf{max}}$ when the upper limit is reached at time $\bar{t} < t$; we cannot express such time $\bar{t}$ *exactly*. This case also disallows *exact* closed-form expressions in the next section where we address the limited KiBaM with randomness. For these reasons we introduce under- and over-approximations of the SoC using simple closed-form expressions. These approximations are employed in Sections 5 to 7 to obtain elegant expressions and also efficient algorithms.

Over-approximation: We circumvent the computation of the time point $\bar{t}$ by moving it to time 0, i.e. to the beginning of the time interval. We assume that the available charge attains a_{max} during the whole time interval and the bound charge evolves *all the time* as captured by $\bar{\mathbf{K}}_t$.

Under-approximation: Dually, we move the time point $\bar{t}$ to time T, i.e. to the end of the time interval. We assume that the charging current has such value $I^{\rightarrow}$ (which is a weaker charging current than I) that causes the available charge to reach a_{max} exactly at the end of the interval. The battery thus evolves *all the time* as captured by $\mathbf{K}_{t,I^{\rightarrow}}$. Expressing $I^{\rightarrow}$ is discussed below.

▶ **Example 17.** Let us illustrate both approximations on the same situation as in Example 14. For the under-approximation (on the left), in the interval $[40, 55]$, we apply $I^{\rightarrow} \approx -432.5$ instead of $I = -600$ so that the available charge reaches 9000 exactly at $t = 55$. From here on, the SoC is in both components lower than the SoC from the previous figure.



For the over-approximation (on the right), in the interval $[40, 55]$, we intuitively apply a load $I \to -\infty$ so that the available charge reaches 9000 exactly at $t = 40$. Since the diffusion is finite, the available charge stays at its limit until $t = 55$ while the bound charge evolves according to $\bar{\mathbf{K}}$. From this point on, the SoC is in both components higher than the SoC from the previous figure.

Finally, we need a closed-form solution to the following problem: From an initial SoC $[a; b]$, we want to reach using $\mathbf{K}$ a certain target level of available charge $\bar{a}$ exactly at time T; which current I achieves this? (For the under-approximation above, we instantiate the problem with $\bar{a} = a_{\mathsf{max}}$.)

Formally, we need to find I such that the first component of $\mathbf{K}_{T,I}[a; b]$ equals $\bar{a}$. The resulting current will be denoted by $I^{\rightarrow}_{\bar{a}}[a; b]$. Later, we also need the solution of the same problem for $\mathbf{K}$'s inverse operator $\mathbf{K}^{-1}$. In this case the question is: From an initial available charge level $\bar{a}$, which current I is necessary to exactly reach a SoC $[a; b]$ at time T? More precisely, find I such that the first component of $\mathbf{K}^{-1}_{T,I}[a; b]$ equals $\bar{a}$. We denote the current that solves this problem by $I^{\leftarrow}_{\bar{a}}[a; b]$. From the above equalities we can derive that these currents are indeed unique, and given by

$$I^{\rightarrow}_{\bar{a}}[a; b] = -\frac{\mathrm{q}_a}{\mathrm{s}_a} \cdot a - \frac{\mathrm{r}_a}{\mathrm{s}_a} \cdot b + \frac{\bar{a}}{\mathrm{s}_a},$$

$$I^{\leftarrow}_{\bar{a}}[a; b] = \frac{-\mathrm{r}_b \cdot a + \mathrm{r}_a \cdot b + (\mathrm{q}_a \mathrm{r}_b - \mathrm{r}_a \mathrm{q}_b) \cdot \bar{a}}{\mathrm{r}_a \mathrm{s}_b - \mathrm{s}_a \mathrm{r}_b}.$$

The bound charge attained under $\mathbf{K}_{T,I^{\rightarrow}_{\bar{a}}}$ and $\mathbf{K}^{-1}_{T,I^{\leftarrow}_{\bar{a}}}$ is denoted by $B^{\rightarrow}_{\bar{a}}[a; b]$ and $B^{\leftarrow}_{\bar{a}}[a; b]$, respectively. In the operators $I^{\rightarrow}_{\bar{a}}$, $I^{\leftarrow}_{\bar{a}}$, $B^{\rightarrow}_{\bar{a}}$, $B^{\leftarrow}_{\bar{a}}$, we omit the initial SoC $[a; b]$, if clear from context.

It is now straight-forward to provide operators $\underline{\mathbf{K}}^{\square}_{t,i}$ and $\overline{\mathbf{K}}^{\square}_{t,i}$ that formally characterize the under- and over-approximations from above.

$$\underline{\mathbf{K}}^{\square}_{t,i}[a_0;b_0] \coloneqq \begin{cases} \mathbf{K}_{t,i}[a_0;b_0] & \text{if } a_0, a_t > 0 \wedge a_t \leq a_{\mathsf{max}}, \\ \mathbf{K}_{t,I^{\rightarrow}_{a_{\mathsf{max}}}}[a_0;b_0] & \text{if } a_0, a_t > 0 \wedge a_t > a_{\mathsf{max}}, \\ [0;0] & \text{if } a_0 \leq 0 \vee a_t \leq 0. \end{cases}$$

$$\overline{\mathbf{K}}^{\square}_{t,i}[a_0;b_0] \coloneqq \begin{cases} \mathbf{K}_{t,i}[a_0;b_0] & \text{if } a_0, a_t > 0 \wedge a_t \leq a_{\mathsf{max}}, \\ \bar{\mathbf{K}}_{t}[a_{\mathsf{max}};b_0] & \text{if } a_0, a_t > 0 \wedge a_t > a_{\mathsf{max}}, \\ [0;0] & \text{if } a_0 \leq 0 \vee a_t \leq 0. \end{cases}$$

$\underline{\mathbf{K}}^{\square}_{t,i}$ and $\overline{\mathbf{K}}^{\square}_{t,i}$ are indeed under- and over-approximation of the exact SoC evolution $\mathbf{K}^{\square}$ in the following sense.

▶ **Lemma 18.** *For any SoCs $[a;b]$ and $T > 0$ and $I > 0$, we have*

$$\underline{\mathbf{K}}^{\square}_{T,I}[a;b] \;\leq\; \mathbf{K}^{\square}_{T,I}[a;b] \;\leq\; \overline{\mathbf{K}}^{\square}_{T,I}[a;b].$$

5 Random Limited KiBaM With Recharging

We now return our attention to the challenge of random KiBaM, enriched by capacity limits. We thus assume that the random variables A_t and B_t evolve according to $\mathbf{K}^{\square}_{t,I}$ developed in Section 4. By overloading notation we change (2) to

$$[A_t;B_t] \coloneqq \mathbf{K}^{\square}_{t,I}[A_0;B_0]. \tag{7}$$

We first observe that the joint distribution of $[A_T;B_T]$ may not be absolutely continuous, because positive probability may accumulate in the point $[0;0]$ where the battery is empty and on the line $\{[a_{\mathsf{max}};b] \mid 0 < b < b_{\mathsf{max}}\}$ where the available charge is full. We need a more complex representation of the distribution.

▶ **Definition 19.** A *SoC distribution* is a triple $\langle f, \bar{f}, z \rangle$ where

- f is a joint density over $]0, a_{\mathsf{max}}[\times]0, b_{\mathsf{max}}[$, (the distribution in the "inner" area)
- $\bar{f}$ is a density over $\{a_{\mathsf{max}}\} \times]0, b_{\mathsf{max}}[$, (bound charge distribution along the capacity limit)
- $z \in [0,1]$. (the cumulative probability of depletion)

We say that a SoC distribution $\langle f_t, \bar{f}_t, z_t \rangle$ *represents* random variables $[A_t;B_t]$ if for any measurable $X \subseteq \mathbb{R} \times \mathbb{R}$ we have

$$\mathbf{Pr}\big[[A_t;B_t] \in X\big] = \iint\limits_{[a;b]\in X} f_t(a,b)\,\mathrm{d}a\,\mathrm{d}b \;+ \int\limits_{[a_{\mathsf{max}};b]\in X} \bar{f}_t(b)\,\mathrm{d}b \;+ z_t\,\mathbb{1}_{[0;0]\in X}$$

where $\mathbb{1}_{\varphi}$ denotes the indicator function of a condition φ.

Similarly to Section 3, we assume random load I described by a probability density function g. For random initial SoC $[A_0;B_0]$ represented by a SoC distribution $\langle f_0, \bar{f}_0, z_0 \rangle$ and a given task (T,g) we aim at expressing the resulting SoC $[A_T;B_T]$ using a SoC distribution $\langle f_T, \bar{f}_T, z_T \rangle$.

To be able to express the distribution as integrals over simple closed-form expressions, we resort to under- and over-approximations of the SoC. We will work with $\lfloor d \rfloor$ and $\lceil d \rceil$ as notations for upper, respectively lower bounding SoC distributions, where d abbreviates the three components of the triple $\langle f_T, \bar{f}_T, z_T \rangle$ (i.e. $\lfloor f_T, \bar{f}_T, z_T \rfloor$ and $\lceil f_T, \bar{f}_T, z_T \rceil$). To arrive there, we define

$$[\underline{A}_T;\underline{B}_T] \coloneqq \underline{\mathbf{K}}^{\square}_{T,I}[A_0;B_0] \qquad \text{and} \qquad [\overline{A}_T;\overline{B}_T] \coloneqq \overline{\mathbf{K}}^{\square}_{T,I}[A_0;B_0]$$

respectively, that under-approximate and over-approximate $[A_T;B_T]$ in the following sense.

▶ **Definition 20.** We say that $[\underline{A}_T; \underline{B}_T]$ *under-approximates* $[A_T; B_T]$ *at the upper limit* if

$$\begin{aligned} \mathbf{Pr}\big[[A_T;B_T] \geq [a;b]\big] &= \mathbf{Pr}\big[[\underline{A}_T;\underline{B}_T] \geq [a;b]\big] && \text{for any } [a;b] < [a_{\mathsf{max}};b_{\mathsf{max}}], \\ \mathbf{Pr}\big[[A_T;B_T] \geq [a_{\mathsf{max}};b]\big] &\geq \mathbf{Pr}\big[[\underline{A}_T;\underline{B}_T] \geq [a_{\mathsf{max}};b]\big] && \text{for any } 0 \leq b \leq b_{\mathsf{max}}. \end{aligned}$$

Analogously, $[\overline{A}_T; \overline{B}_T]$ *over-approximates* $[A_T; B_T]$ *at the upper limit* if

$$\begin{aligned} \mathbf{Pr}\big[[A_T;B_T] \geq [a;b]\big] &= \mathbf{Pr}\big[[\overline{A}_T;\overline{B}_T] \geq [a;b]\big] && \text{for any } [a;b] < [a_{\mathsf{max}};b_{\mathsf{max}}], \\ \mathbf{Pr}\big[[A_T;B_T] \geq [a_{\mathsf{max}};b]\big] &\leq \mathbf{Pr}\big[[\overline{A}_T;\overline{B}_T] \geq [a_{\mathsf{max}};b]\big] && \text{for any } 0 \leq b \leq b_{\mathsf{max}}. \end{aligned}$$

This approach, detailed in the rest of this section, provides upper and lower bounds on the risk of battery depletion due to the monotonicity established in Lemma 28.

Behaviour below the upper limit (defining f_T and z_T). We first define a joint density $\underline{f}_T$ over $]-\infty, a_{\mathsf{max}}[\times]-\infty, b_{\mathsf{max}}[$ that exactly describes the behaviour below the upper limit while *ignoring* the lower limit. This allows us to define

$$f_T(a,b) \coloneqq \underline{f}_T(a,b), \qquad \text{and} \qquad z_T \coloneqq \iint_{\mathbb{R}^2_{\leqslant 0}} \underline{f}_T(a,b)\,\mathrm{d}a\,\mathrm{d}b. \tag{8}$$

Note that for this case both, under- and over-approximation, behave equally, as indicated in Definition 20.

The intricate part in expressing $\underline{f}_T$ is to describe how the SoC evolves away from the upper limit to the area below the upper limit when the level of available charge is decreasing. For each SoC $[a';b']$ below the upper limit we need to find out what SoC of the form $[a_{\mathsf{max}};b]$ under what load i (such that $i > \bar{I}(b)$) would evolve in time T exactly into $[a';b']$, i.e $\mathbf{K}^{-1}_{T,i}[a';b'] = [a_{\mathsf{max}};b]$. By definition, this is the case when using load $I^{\leftarrow}_{a_{\mathsf{max}}}[a';b']$ which results in a bound charge $b = B^{\leftarrow}_{a_{\mathsf{max}}}[a';b']$. The Jacobian determinant of the map $[a;b] \mapsto \big[B^{\leftarrow}_{a_{\mathsf{max}}}; I^{\leftarrow}_{a_{\mathsf{max}}}\big]$ is easily derived to be $1/(\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b)$ and is constant in the SoC and the load.

Finally, we can express the joint density $\underline{f}_T$ for any $a < a_{\mathsf{max}}$ and $b < b_{\mathsf{max}}$ as

$$\underline{f}_T(a,b) \;=\; \bar{f}_0\big(B^{\leftarrow}_{a_{\mathsf{max}}}\big) \cdot \frac{1}{|\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b|} \cdot g\big(I^{\leftarrow}_{a_{\mathsf{max}}}\big) + \mathrm{e}^{kT} \int_{-\infty}^{I^{\leftarrow}_{a_{\mathsf{max}}}} f_0\Big(\mathbf{K}^{-1}_{T,i}[a;b]\Big) \cdot g(i)\,\mathrm{d}i. \tag{9}$$

The second summand in (9) comes from the density f_0 of the inner area by the standard unlimited KiBaM. Ranging over all loads i, it integrates the density f_0 of such points $[a_i;b_i]$ that satisfy $\mathbf{K}_{T,i}[a_i;b_i] = [a;b]$, i.e. $[a_i;b_i] = \mathbf{K}^{-1}_{T,i}[a;b]$. Lemma 5 again guarantees that no limits are crossed in the meantime. The first summand comes from $\bar{f}_0$, due to discharging the battery down from the capacity limit as discussed above.

Behaviour on the upper limit (defining $\bar{f}_T$). As indicated in Definition 20, we resort for the upper limit to approximations of the charge.

Under-approximation: We define the under-approximation of the density for $0 \leq b \leq b_{\mathsf{max}}$ by

$$\bar{f}_T(b) \;=\; \bar{f}_0\big(\bar{b}^{-1}\big) \cdot G\big(\bar{I}\big(\bar{b}^{-1}\big)\big) \cdot \mathrm{e}^{ckT} \tag{10}$$

$$+\; \bar{f}_0\big(B^{\leftarrow}_{a_{\mathsf{max}}}\big) \cdot \Big[G\big(I^{\leftarrow}_{a_{\mathsf{max}}}\big) - G\big(\bar{I}\big(B^{\leftarrow}_{a_{\mathsf{max}}}\big)\big)\Big] \cdot \left|\frac{-\mathrm{s}_a}{\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b}\right| \tag{11}$$

$$+\; \int_{-\infty}^{\infty} f_0\big(a, B^{\leftarrow}_a\big) \cdot G\big(I^{\leftarrow}_a\big)\,\mathrm{d}a \cdot \left|\frac{-\mathrm{s}_a}{\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b}\right| \tag{12}$$

Let us go through this expression line by line.

In the first summand (10), $\bar{b}^{-1}$ is such that $\bar{\mathbf{K}}_T\big[a_{\mathsf{max}};\bar{b}^{-1}\big] = [a_{\mathsf{max}};b]$. Thus $\bar{b}^{-1} \coloneqq \bar{b}_T^{-1}(b) = \mathrm{e}^{ckT}\cdot b - (\mathrm{e}^{ckT}+1)\cdot b_{\mathsf{max}}$ where the function $\bar{b}_T$ is defined in Definition 5. This summand is taken into account only for charging currents that cover the diffusion (i.e. $i \leq \bar{I}\big(\bar{b}^{-1}\big)$) so that the battery evolves along the capacity limit as expressed by Lemma 13. The integration over this range of loads can be directly expressed using the *cumulative density function (cdf)* G of the load. Technically, we again apply the transformation law for random variables.
By the second summand (11), we address the case where the diffusion in the state $[a_{\mathsf{max}};b]$ is stronger than the charging current. The available charge thus leaves its limit in the beginning. Let us assume that before time T it again hits the upper capacity in some state $[a_{\mathsf{max}};b']$. We are not able to express b using a closed-form expression over b' as discussed in Section 4. As a result, we cannot "move" the density from b to b'. We thus under-approximate the bound charge by assuming that the available charge hits its upper limit again exactly at time T by charging the battery with $I^{\rightarrow}_{a_{\mathsf{max}}}$ instead, just as shown in Example 17. Let us shortly outline the derivation. The mapping for the transformation law is $b \mapsto B^{\rightarrow}_{a_{\mathsf{max}}}[a_{\mathsf{max}};b]$. Its inverse is simply $b \mapsto B^{\leftarrow}_{a_{\mathsf{max}}}[a_{\mathsf{max}};b]$ with Jacobian determinant $-\mathrm{s}_a/(\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b)$. The transformation law yields a density at time T of

$$\bar{f}_0\big(B^{\leftarrow}_{a_{\mathsf{max}}}[a_{\mathsf{max}};b]\big)\cdot|-\mathrm{s}_a/(\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b)|.$$

Then we integrate over all charging currents i that are powerful enough to reach the limit ($i \leq I^{\leftarrow}_{a_{\mathsf{max}}}[a_{\mathsf{max}};b]$) yet not too powerful to leave the upper limit in the meantime, i.e. $i > \bar{I}\big(B^{\leftarrow}_{a_{\mathsf{max}}}[a_{\mathsf{max}};b]\big)$. The integral over the resulting range equals $G\big(I^{\leftarrow}_{a_{\mathsf{max}}}\big) - G\big(\bar{I}\big(B^{\leftarrow}_{a_{\mathsf{max}}}\big)\big)$.
The third summand (12) comes from the density f_0 of the inner area and under-approximates the bound charge similarly to the second summand. If the available charge of the battery reaches its capacity limit *before* time T, we assume that it reaches it exactly at time T by underestimating the charging current with $I^{\rightarrow}_{a_{\mathsf{max}}}$. For the derivation, we define a map $\mathcal{K}_T : [a;b;i] \mapsto \big[a;B^{\rightarrow}_{a_{\mathsf{max}}}[a;b];i\big]$ (it is an identity in the first and the third component to be injective) and apply the transformation law of random variables. The inverse $\mathcal{K}_T^{-1}$ and its Jacobian determinant is

$$\mathcal{K}_T^{-1} : [a;b;i] \mapsto [a;B^{\leftarrow}_a[a_{\mathsf{max}};b];i] \qquad \text{and} \qquad \mathbf{det}J_{\mathcal{K}_T^{-1}} = -\mathrm{s}_a/(\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b).$$

The density h_T over the co-domain of $\mathcal{K}_T$ is obtained by the transformation law as

$$h_T[a;b;i] = h_0\big(\mathcal{K}_T^{-1}[a;b;i]\big)\cdot\left|\frac{-\mathrm{s}_a}{\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b}\right| = f_0(a,B^{\leftarrow}_a[a_{\mathsf{max}};b])\cdot g(i)\cdot\left|\frac{-\mathrm{s}_a}{\mathrm{r}_a\mathrm{s}_b - \mathrm{s}_a\mathrm{r}_b}\right|$$

where the density h_0 equals a product of the densities f_0 and g because of independence of SoC $[a;b]$ and load i. Marginalizing away a and i (using $G(I^{\leftarrow}_a)$ to integrate over all currents necessary to reach the upper limit) gives us the subdensity from the third summand.

Over-approximation: The over-approximation bases on similar building blocks and equals:

$$\bar{f}_T(b) \quad = \quad \bar{f}_0\big(\bar{b}^{-1}\big)\cdot G\big(I^{\rightarrow}_{a_{\mathsf{max}}}\big)\cdot \mathrm{e}^{ckT} \quad + \int_{-\infty}^{\infty} f_0\big(a,\bar{b}^{-1}\big)\cdot G\big(I^{\rightarrow}_{a_{\mathsf{max}}}\big)\,\mathrm{d}a\cdot\mathrm{e}^{ckT} \tag{13}$$

The first summand in (13) treats the contribution of the density $\bar{f}_0$ from the point $\bar{b}^{-1}$ as defined above. We assume the density evolves as indicated by $\bar{\mathbf{K}}$ whenever $\mathbf{K}$ would result in $a_T \geq a_{\mathsf{max}}$ (i.e. if the current is stronger than $I^{\rightarrow}_{a_{\mathsf{max}}}\big[a_{\mathsf{max}};\bar{b}^{-1}\big]$). This is an over-approximation for the charging currents such that $\bar{I}\big(\bar{b}^{-1}\big) < i < I^{\rightarrow}_{a_{\mathsf{max}}}\big[a_{\mathsf{max}};\bar{b}^{-1}\big]$, i.e. for charging currents that are not strong enough to stay at a_{max} for the whole time but that are stronger than what is needed for $\mathbf{K}$ to return to a_{max} at time T.

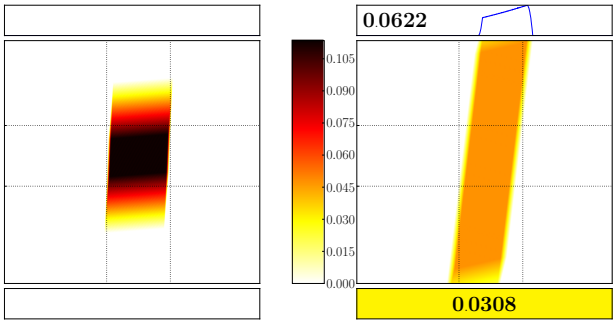
The second summand in (13) comes from the density f_0 of the inner area. Again, whenever $\mathbf{K}$ would result in $a_T > a_{\max}$ (i.e. when the charging current is stronger than $\underline{I}^{\rightarrow}_{a_{\max}}\left[a; \bar{b}^{-1}\right]$), we assume that the upper limit is reached immediately and further evolves by $\overline{\mathbf{K}}$, thus justifying the argument $I^{\rightarrow}_{a_{\max}}$ to appear in the integral. This results in an over-approximation for any such SoC.

The derivation of both summands in principle uses the mapping $[a; b; i] \mapsto \left[a; \bar{b}_T(b); i\right]$ (for the first summand, think of a being the constant $a_{\max}$). The transformation law and marginalizing away a (for the second summand) and i (integration over the corresponding range is again expressed using the cdf G) as in the derivations for the under-approximation provide the result.

We finally obtain the following result.

▶ **Lemma 21.** *Let (T, g) be a task, $\langle f_0, \bar{f}_0, z_0 \rangle$ represent $[A_0; B_0]$ and the induced SoC distributions $\lfloor f_T, \bar{f}_T, z_T \rfloor$ and $\lceil f_T, \bar{f}_T, z_T \rceil$ represent $[\underline{A}_T; \underline{B}_T]$ and $[\overline{A}_T; \overline{B}_T]$. Then $[\underline{A}_T; \underline{B}_T]$ under-approximates $[A_T; B_T]$ and $[\overline{A}_T; \overline{B}_T]$ over-approximates $[A_T; B_T]$ at the upper limit.*

▶ **Example 22.** Based on Lemma 21, we can under-approximate the SoC of the random battery from our second running example for *battery limits* $[0, 10]$. We consider the same tasks, $(20, g)$ on the left and $(60, g)$ on the right.



The bounded area of the joint density f_T is depicted by the largest box. In the small box above we display the density $\bar{f}_T$ at the capacity limit $a_{\max}$. The numbers above and below are the probabilities of available charge being full and empty, respectively (the color below corresponds to the probability).

▶ **Lemma 23** (Probability of $\mathbf{K}^{\square}$-powering a task)**.** *A battery with SoC distribution $\langle f_0, \bar{f}_0, z_0 \rangle$ $\mathbf{K}^{\square}$-powers with probability $p > 0$ a task (T, g) if and only if $z_T < p$.*

6 Markov Task Process

So far, we have only discussed execution of one task with fixed duration and random load. In this section, we give a discrete-time Markov model that randomly generates tasks that we call a *Markov task process* (MTP). The formalism is closely inspired by stochastic task graph models [34] or data-flow formalisms such as SDF [26] or SADF [38]. In SDF, task durations are deterministic, and thus directly supported in our framework. In SADF, durations are in general governed by discrete probability distributions, which can be translated into our framework at the price of a larger state space. We will briefly explain informally how to translate timed versions of SDF as well as SADF to MTPs, after formally introducing the latter.

▶ **Definition 24.** A *Markov task process (MTP)* is a tuple $\mathcal{M} = (S, P, \pi, \Delta, \mathbf{g})$ where S is a finite set of tasks, $P : S \times S \to [0, 1]$ is a transition probability matrix, π is an initial probability distribution over S, $\Delta : S \to \mathbb{N} \setminus \{0\}$ assigns to each task an positive integer time duration, and $\mathbf{g}$ assigns to each task a probability density function of the load.

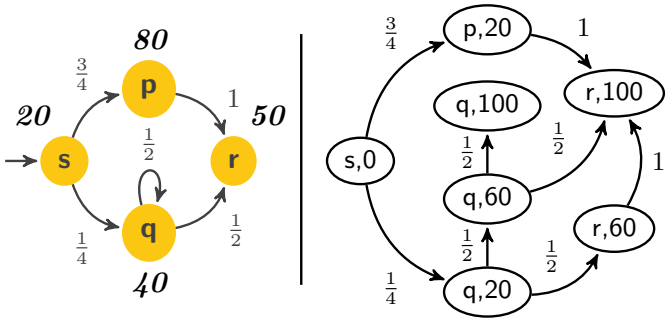


Figure 2 An MTP model on the left (with Δ depicted next to states) and its induced graph for $T = 100$ on the right.

An example of an MTP is depicted in Figure 2. Intuitively, a Markov task process $\mathcal{M}$ together with an initial distribution of the SoC given by $\langle f_0, \bar{f}_0, z_0 \rangle$ behaves as follows. First, an initial SoC $[a_0; b_0]$ of the battery and an initial task $s_0 \in S$ are chosen independently at random according to $\langle f_0, \bar{f}_0, z_0 \rangle$, and π, respectively. Then, the load i_0 in task s_0 is picked randomly according to $\mathbf{g}(s_0)$. After the battery is strained by the load i_0 for $\Delta(s_0)$ time units, the process moves into a random successor task s_1 (where any s_1 is chosen with probability $P(s_0, s_1)$). Here, the load i_1 is randomly chosen and so on.

Formally, $\mathcal{M}$ and $\langle f_0, \bar{f}_0, z_0 \rangle$ induce a probability measure $\mathbf{Pr}$ over samples of the form $\omega = \big[[a_0; b_0]; (s_k, i_k)_{k=0}^{\infty}\big]$ where the first component is the initial SoC of the battery and the second component describes an infinite execution of $\mathcal{M}$. Here, each s_j is the j-th task and i_j is the load that is put on the battery for $\Delta(s_j)$ time units while performing s_j. For a given $T \in \mathbb{R}_{\geqslant 0}$, the SoC of the battery at time T is expressed by random variables A_T, B_T that are for any $\omega = \big[[a_0; b_0]; (s_k, i_k)_{k=0}^{\infty}\big]$ defined as

$$\begin{bmatrix} A_T(\omega) \\ B_T(\omega) \end{bmatrix} \coloneqq \mathbf{K}^{\square}_{\Delta', i_n} \circ \mathbf{K}^{\square}_{\Delta_{n-1}, i_{n-1}} \circ \cdots \circ \mathbf{K}^{\square}_{\Delta_0, i_0} \begin{bmatrix} a_0 \\ b_0 \end{bmatrix}$$

where each Δ_j stands for $\Delta(s_j)$, and n is the minimal number such that the n-th task is not finished before T, i.e. $\Delta_n > \Delta'$ where $\Delta' \coloneqq T - \sum_{j=0}^{n-1} \Delta_j$.

▶ **Definition 25.** We say that a battery with a SoC $\langle f_0, \bar{f}_0, z_0 \rangle$ *powers with probability* $p > 0$ a system $\mathcal{M}$ for time T if

$$\mathbf{Pr}[A_T > 0] \geq p.$$

In order to under-approximate the probability that $\mathcal{M}$ is powered for a given time, we need to symbolically express the distribution of

$$\begin{bmatrix} \underline{A}_T(\omega) \\ \underline{B}_T(\omega) \end{bmatrix} \coloneqq \underline{\mathbf{K}}^{\square}_{\Delta', i_n} \circ \underline{\mathbf{K}}^{\square}_{\Delta_{n-1}, i_{n-1}} \circ \cdots \circ \underline{\mathbf{K}}^{\square}_{\Delta_0, i_0} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}$$

where we just replace $\mathbf{K}^{\square}$ with $\underline{\mathbf{K}}^{\square}$. Analogously, for an over-approximation we use $\overline{\mathbf{K}}^{\square}$ instead.

$$\begin{bmatrix} \overline{A}_T(\omega) \\ \overline{B}_T(\omega) \end{bmatrix} \coloneqq \overline{\mathbf{K}}^{\square}_{\Delta', i_n} \circ \overline{\mathbf{K}}^{\square}_{\Delta_{n-1}, i_{n-1}} \cdots \circ \overline{\mathbf{K}}^{\square}_{\Delta_0, i_0} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}.$$

We present an algorithm that builds upon the previous results.

Expressing the distribution of $[\underline{A_T}; \underline{B_T}]$ and $\lceil \overline{A_T}; \overline{B_T} \rceil$. Let us fix an **input** MTP $\mathcal{M} = (S, P, \pi, \Delta, \mathbf{g})$, SoC distribution $\langle f_0, \bar{f}_0, z_0 \rangle$ that represents $[A_0; B_0]$, and time $T > 0$. We consider the joint distribution of under- / over-approximation of the SoC and the MTP. Intuitively, we split the SoC distribution into under- and over-approximating subdistributions and move them along the paths of $\mathcal{M}$ according to the probabilistic branching of the MTP. We notice that we do not need to explore all exponentially many paths; when two paths visit the same state at the same moment, we can again merge the two subdistributions. This process is formalized by the following graph and a procedure how to propagate the distribution through the graph.

For a given MTP $\mathcal{M}$ we define a directed acyclic graph (V, E) over $V = S \times \{0, 1, \dots, \lfloor T \rfloor, T\}$ such that there is an edge from a vertex (s, t) to a vertex (s', t') if $P(s, s') > 0$, $t < t'$, and $t' = \min\{t + \Delta(s), T\}$. Further, let (V', E') be the graph obtained from (V, E) by removing vertices that are not reachable from any $(s, 0)$ with $\pi(s) > 0$ (see Figure 2).

1. We label each vertex of the form $(s, 0)$ where $\pi(s) > 0$ by the pair of equal initial subdistributions

$$\lfloor f, \bar{f}, z \rfloor \coloneqq \langle f_0, \bar{f}_0, z_0 \rangle \cdot \pi(s) \qquad \text{and} \qquad \lceil f, \bar{f}, z \rceil \coloneqq \langle f_0, \bar{f}_0, z_0 \rangle \cdot \pi(s)$$

where the multiplication is to be understood componentwise.

2. We repeat the following steps as long as possible.

a. For each vertex (s, t) labeled by $\lfloor f, \bar{f}, z \rfloor$ and $\lceil f, \bar{f}, z \rceil$, we obtain $\lfloor f_\Delta, \bar{f}_\Delta, z_\Delta \rfloor$ and $\lceil f_\Delta, \bar{f}_\Delta, z_\Delta \rceil$ by Lemma 21 for a task $(\Delta, \mathbf{g}(s))$ where $\Delta \coloneqq \min\{\Delta(s), T - t\}$. Then we label i-th outgoing *edge* of (s, t) leading to some (s', t') by

$$\lfloor f^i, \bar{f}^i, z^i \rfloor \coloneqq \lfloor f_\Delta, \bar{f}_\Delta, z_\Delta \rfloor \cdot P(s, s') \qquad \text{and} \qquad \lceil f^i, \bar{f}^i, z^i \rceil \coloneqq \lceil f_\Delta, \bar{f}_\Delta, z_\Delta \rceil \cdot P(s, s').$$

b. For each vertex (s, t) such that its k ingoing edges are labelled by $\lfloor f^i, \bar{f}^i, z^i \rfloor$ and $\lceil f^i, \bar{f}^i, z^i \rceil$ for $i = 1, \dots, k$, we label (s, t) by

$$\lfloor f, \bar{f}, z \rfloor \coloneqq \sum_{i=1}^{k} \lfloor f^i, \bar{f}^i, z^i \rfloor \qquad \text{and} \qquad \lceil f, \bar{f}, z \rceil \coloneqq \sum_{i=1}^{k} \lceil f^i, \bar{f}^i, z^i \rceil$$

where the summation is again to be interpreted componentwise.

Finally, let i-th of all n vertices of the form $(s, T) \in V'$ be labelled by $\lfloor f, \bar{f}, z \rfloor_i$ and $\lceil f, \bar{f}, z \rceil_i$. The **output** distributions that represent $[\underline{A}_T; \underline{B}_T]$ and $\lceil \overline{A}_T; \overline{B}_T \rceil$ respectively are

$$\lfloor f_T, \bar{f}_T, z_T \rfloor \coloneqq \sum_{i=1}^{n} \lfloor f, \bar{f}, z \rfloor_i \qquad \text{and} \qquad \lceil f_T, \bar{f}_T, z_T \rceil \coloneqq \sum_{i=1}^{n} \lceil f, \bar{f}, z \rceil_i.$$

We naturally arrive at the following theorem.

▶ **Theorem 26.** *A battery with SoC distribution $\langle f_0, \bar{f}_0, z_0 \rangle$ $\mathsf{K}^\square$-powers a system $\mathcal{M}$ for time T with probability at least $1 - \underline{z}_T$ and at most $1 - \overline{z}_T$, where $\underline{z}_T$ and $\overline{z}_T$ are the depletion probabilities of the densities representing $[\underline{A}_T; \underline{B}_T]$ and $\lceil \overline{A}_T; \overline{B}_T \rceil$, respectively.*

This theorem relies on the simple observation that an underapproximation of the SoC is an overapproximation of the depletion probability.

▶ Remark (on complexity). As indicated in the beginning of this section, we do not need to track all exponentially many paths through the MTP up to time T. In fact, in the algorithm above, once we have computed the subdistributions on the left hand side, we can discard the subdistribution on the right hand side of the assignments. Since the task durations Δ are natural numbers, the amount of subdistributions we need to track simultaneously is bounded by $|S| \cdot D$ where D is the smallest common multiple of all the task durations. D always exists since all task durations are strictly positive.

Translating timed SDF graphs to equivalent MTPs. As mentioned above, some well known formalisms can be translated to MTPs, among them the timed version of *Synchronuous Data Flow (SDF)* [26] and some *Scenario-Aware Data Flow (SADF)* [38] flavors. We will demonstrate informally how to translate a timed SDF graph (SDFG) to an equivalent MTP.

SDF is a widely used formalism for modelling and analysing networks of deterministic sequential processes along with their resource budget. Processes, called *actors* communicate via consuming and producing tokens (data elements) from their incoming and to their outgoing unbounded channels. Whenever an actor is activated it spawns a new active *instance*. The number of tokens an instance consumes and eventually produces is fixed a priori. The timed version additionally annotates each actor a with a constant execution time $e(a) \in \mathbb{N} \setminus \{0\}$, representing how much time passes between consumption and production of tokens. We furthermore associate a distribution $L(a)$ over loads with each actor a to reason about energy consumption.

The semantics of an SDFG execution is a finite *Labelled Transition System (LTS)* over its configurations, which can be extended to an MTP on the same state space of configurations with Dirac transitions and additional annotations concerning load and sojourn times.

An SDF configuration records

(i) a vector v representing the number of tokens in each channel,
(ii) a set $\mathcal{A}$ collecting active actor instances a_i of any actor a and
(iii) the residual execution time of each running instance as a map r.

Transitions between states are of three natures:

start a: A new instance a_i of actor a with $r(a_i) = e(a)$ is added to $\mathcal{A}$, provided the input channels contain enough tokens, which are thereby consumed;

end a: An actor instance a_i is removed from $\mathcal{A}$ when its residual execution time $r(a_i)$ is 0. Thereby output tokens are produced according to a's output channels;

time t: Under the precondition that no **start** or **end** transitions are possible, an amount of time t passes corresponding to the minimum of the residual times, thereby decreasing the residual execution times of every active actor instance accordingly.

The precondition of **time**-type transitions implies that the LTS is free of nondeterminism between **time** and **start**/**end** transitions. The nondeterminism among **start**/**end** transitions is irrelevant thanks to the *diamond property*. This means that for each state s there is a unique *final* state s' such that each maximal sequence of **start**/**end** transitions from s ends up in s'. On each such diamond (set of states reachable from s) no time passes, thus it has no effect on the battery. As the first step in defining the MTP, we transform the LTS by collapsing each diamond into its final state. As a result, the LTS becomes deterministic with only **time** transitions remaining. If the starting state was part of a diamond collapsed to a state s', this state becomes the initial state of the transformed LTS.

The reachable part of the LTS induces an MTP $\mathcal{M} = (S, P, \pi, \Delta, \mathbf{g})$ as follows:

- The state space S is defined as the reachable states of the LTS,
- The initial probability distribution π is Dirac in the initial state of the SDFG,
- $\Delta(s)$ for $s \in S$ is the residual time according to the transition of type **time** leaving s.
- $\mathbf{g}(s)$ for $s = (v, \mathcal{A}, r) \in S$ is the convolution of the $L(a)$ for each $a_i \in \mathcal{A}$.
- $P(s, s')$ is 1 if there is a **time** transition from s to s', and 0, otherwise.

SADF extends SDF to discrete execution time distributions and scenarios (with subscenarios probabilistically chosen through discrete-time Markov Chains). An extension of the above is relatively intuitive, but technically involved. However, since the semantics of such SADF graphs under self-timed executions are *Timed Probabilistic Systems (TPS)* with the diamond property for actions [39], an analogous approach to the above can be formulated.

7 Approximating Random Limited KiBaM With Recharging

After approaching the problems from the theoretical side, we take the practical view in this section. We want to be able to compute the probability to power a system $\mathcal{M}$ for a given time T in practice. As this is only possible approximatively, we want to obtain provably correct lower and upper bounds on this probability.

The crucial step in the symbolic algorithm from Section 6 is the following: for a fixed initial SoC distribution $\langle f_0, \bar{f}_0, z_0 \rangle$, compute the SoC distribution $\langle f_T, \bar{f}_T, z_T \rangle$ after powering a task (T, g). First, we implemented the symbolic continuous solution developed in Sections 3 and 5 in a high-level computational language Octave. This way, we performed numerical integration only over the resulting complete expression describing the SoC distribution at time T. This showed up to be practical only up to sequences of a handful of tasks. Thus, we targeted discretisation of the SoC space and of the load distributions. In contrast to general approaches [35], we are able to give much tighter (a posteriori) error bounds.

The discretisation algorithm. We need to assume that each load distribution g is supported only on a bounded interval. This is no real restriction due to the obvious physical limits of battery load.

The idea is simple. We approximate each SoC distribution over $[0, a_{\mathsf{max}}] \times [0, b_{\mathsf{max}}]$ by a discrete distribution μ over a regular grid $[0, \delta, 2\delta, \dots, a_{\mathsf{max}}] \times [0, \delta, 2\delta, \dots, b_{\mathsf{max}}]$, for any fixed $\delta > 0$ that divides the maximum capacities a_{max} and b_{max} into $K \coloneqq a_{\mathsf{max}}/\delta$ and $L \coloneqq b_{\mathsf{max}}/\delta$ steps.[1]

For a fixed initial SoC distribution μ and task (T, g), we define an under-approximated target distribution $\underline{\mu}$ and an over-approximated target distribution $\overline{\mu}$ for each point $[k\delta; l\delta]$ as follows. We first over-approximate and under-approximate the density g by discrete distributions $\overline{g}$ and $\underline{g}$ supported on multiples of $\delta_g > 0$.[2] Then we set

$$\underline{\mu}[k\delta; l\delta] \coloneqq \sum_i \overline{g}(i\delta_g) \cdot \sum \left\{ \mu[k'\delta; l'\delta] \;\middle|\; \underline{\mathbf{K}}^{\square}_{t,i}[k'\delta; l'\delta] \in [k\delta, (k+1)\delta[\times [l\delta, (l+1)\delta[\right\},$$

$$\overline{\mu}[k\delta; l\delta] \coloneqq \sum_i \underline{g}(i\delta_g) \cdot \sum \left\{ \mu[k'\delta; l'\delta] \;\middle|\; \overline{\mathbf{K}}^{\square}_{t,i}[k'\delta; l'\delta] \in](k-1)\delta, k\delta] \times](l-1)\delta, l\delta] \right\}.$$

Intuitively, in the definition above we apply the *deterministic* KiBaM operator $\mathbf{K}^{\square}$ to each possible load and round the result to the closest multiples of δ. The results are weighted by the approximation of the load distribution. The direction of the approximation determines the direction of the approximation of g, the direction of approximation of the function $\mathbf{K}^{\square}_{t,i}$, and the direction of rounding the result.

Correctness of the approximations. When plugging the under- and over-approximation of each step for given δ to the algorithm in Section 6, we obtain the overall algorithm to compute under- and over-approximations $\underline{\mu}_T$ and $\overline{\mu}_T$ of the SoC distribution after powering a given system $\mathcal{M}$ for a given time horizon $T > 0$. Let $\left[\underline{A}^{\delta}_T; \underline{B}^{\delta}_T\right]$ and $\left[\overline{A}^{\delta}_T; \overline{B}^{\delta}_T\right]$ denote random variables distributed according to $\underline{\mu}_T$ and $\overline{\mu}_T$.

[1] Here we assume that c is rational thus allowing to find arbitrarily small such $\delta > 0$. This is assumed purely for presentation purposes, as for the under-approximation the capacity limits can be arbitrarily decreased; for the over-approximation analogously increased.

[2] The over-approximation of the load assigns the integral of g over $[k \cdot \delta_g, (k+1) \cdot \delta_g]$ to the point k, the under-approximation to the point $k+1$.

▶ **Theorem 27.** *Let* $[A_0; B_0]$ *be the initial SoC, represented by* $\langle f_0, \bar{f}_0, z_0 \rangle$, $\delta > 0$, $\mathcal{M}$ *be an MTP and* $T > 0$ *be a time horizon. For any SoC* $[a; b]$ *on the grid (i.e. equal to some* $[k\delta; \ell\delta]$*) we have*

$$\mathbf{Pr}\big[\big[\underline{A}^{\delta}_{T}; \underline{B}^{\delta}_{T}\big] \geq [a; b]\big] \;\leq\; \mathbf{Pr}\big[[A_T; B_T] \geq [a; b]\big] \;\leq\; \mathbf{Pr}\big[\big[\overline{A}^{\delta}_{T}; \overline{B}^{\delta}_{T}\big] \geq [a; b]\big].$$

The theorem relies on Lemma 18 and one additional observation. The KiBaM evolution has one fundamental property: for any fixed time and load, it is monotonic with respect to starting SoC.

▶ **Lemma 28.** *For any two SoCs* $[a; b]$, $[a'; b']$ *as well as* $T > 0$ *and* $I > 0$, *we have*

$$[a; b] \leq [a'; b'] \implies \mathsf{K}^{\square}_{T,I}[a; b] \leq \mathsf{K}^{\square}_{T,I}[a'; b'].$$

This property is crucial for correctness and not found in general systems studied in [35]. It implies that a sequence of under-approximations is still an under-approximation, and dually for over-approximations.

8 The Random KiBaM In Practice

In this section, we apply the results established in the previous sections in a concrete scenario. The problem is inspired by experiments currently being carried out with an earth orbiting nano satellite, the GomX–1 [20].

Satellite. GomX–1 [20] is a Danish two-unit CubeSat mission launched in November 2013 to perform research and experimentation in space related to Software Defined Radio (SDR) with emphasis on receiving ADS-B signals from commercial aircraft over oceanic areas. As a secondary payload the satellite flies a NanoCam C1U color camera for earth observation experimentation. Five sides are covered with NanoPower P110 solar panels, and the power system NanoPower P31u holds a 7.4V Li-Ion battery of capacity 5000 mAh. GomX–1 uses a radio amateur frequency for transmitting telemetry data, making it possible to receive the satellite data with low-cost infrastructure anywhere on earth. The mission is developed in collaboration between GomSpace ApS, DSE Airport Solutions and Aalborg University, financially supported by the Danish National Advanced Technology Foundation. The empirical studies carried out with GomX–1 serve as a source for parameter values and motivate the scenario described in the remainder of the paper. We use the following data collected from extensive in-flight telemetry logs.

- One orbit takes 99 minutes and is nearly polar;
- The battery capacity is $\mathsf{cap} = 5000$ mAh;
- During 4 to 7 out of on average 15 orbits per day, communication with the base station takes place. The load induced by communication is roughly 400 mA. The length of the communication depends on the distance of the pass of the satellite to the base station and varies between 5 and 15 minutes;
- In each communication, the satellite can receive instructions on what activities to perform next. This influences the subsequent background load. Three levels of background load dominate the logs, with average loads at 250 mA, 190 mA, and 90 mA. These background loads subsume the power needed for operating the respective activities, together with basic tasks such as sending beacons every 10 seconds;
- Charging happens periodically, and spans around 2/3rd of the orbiting time. Average charge power is 400 mA.

The above empirical observations determine the base line of our modelling efforts, which interprets the statistical data as being of stochastic nature. We make the following assumptions:

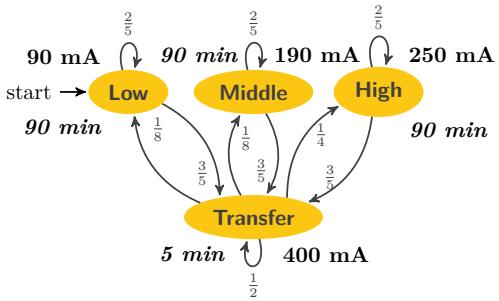


Figure 3 Markov task process of the load on the satellite. All load distributions are normal with mean depicted next to the states and with standard deviation 5. This load is superposed with a strictly periodic load modelling charge by solar power infeed.

- We assume constant battery temperature. The factual temperature of the orbiting battery oscillates between −8 and 25 degree Celsius on its outside. There is the (currently unused) on-board option to heat the battery to nearly constant temperature. Using an on-off controller, this would lead to another likely nearly periodic load on the battery, well in the scope of what our model supports.
- A constant charge from the solar panels is assumed when exposed to the sun. The factual observed charge slowly decays. This is likely caused by the fact that solar panels operate better at lower temperature (opposite to batteries), but heat up quickly when coming out of eclipse.
- We assume a strictly periodic charging behavior. The factual charging follows a more complicated pattern determined by the relative position of sun, earth and satellite. There is no fundamental obstacle to calculate and incorporate that pattern.
- We assume a uniform initial charge between 70% and 90% of full capacity with identical bound and available charge. Since the satellite needs to be switched off for transportation into space, assuming an equilibrated battery is valid. Being a single experiment, the GomX–1 had a particular initial charge (though unknown). The charge of the orbiting battery can only be observed indirectly, by the voltage sustained.
- We assume that the relative distance to a base station is a random quantity, and thus interpret several of the above statistics probabilistically. In reality, the position of the base station for GomX–1 is at a particular fixed location (Aalborg, Denmark). Our approach can either be viewed as a kind of probabilistic abstraction of the relative satellite position and uncertainty of signal transmission, or it can be seen as reflecting that base stations are scattered around the planet. This especially would be a realistic in scenarios where satellite-to-satellite communication is used.
- We assume that the satellite has no protection against battery depletion. In reality, the satellite has 2 levels of software protection, activated at voltage levels 7.2 and 6.5, respectively, backed up by a hardware protection activated at 6 V. In these protection modes, various non-mission-critical functionality is switched-off. Despite omitting such power-saving modes, we still obtain conservative guarantees on the probability that the battery powers the satellite.

Satellite model. According to the above discussion, the load on the satellite is the superposition of two piecewise constant loads.

- A probabilistic load reflecting the different operation modes, modeled by a Markov task process $\mathcal{M}$ as depicted in Figure 3.
- A strictly periodic charge load alternating between 66 minutes at −400 mA, and the remaining 33 minutes at 0 mA.

One can easily express the charging load as another independent Markov task process (where all probabilities are 1) and consider the sum load generated by these two processes in parallel (methods in Section 6 adapt straightforwardly to this setting).

The KiBaM in our model has following parameters:

- The ratio of the available charge $c = 1/2$ (artificially chosen value as parameters fitted by experiments on similar batteries strongly vary [42, 25]);
- The diffusion rate $p = 0.0006$ per minute (we decreased the value reported by experiments [25] by a factor of 4 because of the low average temperature in orbit, 3.5°C, and the influence of the Arrhenius equation [27]).

Implementation aspects. Our implementation is done in C++. We used $K = 1200, 600, 300$ and 150 for the experiments with the batteries of capacity 5000 mAh, 2500 mAh, 1250 mAh and 625 mAh, respectively to guarantee equal relative precision. All the experiments have been performed on a machine equipped with an Intel Core i5-2520M CPU @ 2.50GHz and 4GB RAM. All values occuring are represented and calculated with standard IEEE 754 double-precision binary floating-point format except for the values related to the battery being depleted where we use arbitrary precision arithmetic (as this number keeps accumulating grid values that are of much lower order of magnitude). The number of subdistributions that must be kept track of simultaneously, turned out to be no larger than 54.

Model evaluation. We performed various experiments with this model, to explore the random KiBaM technology. We here report on five distinct evaluations, demonstrating that valuable insight into the model can be obtained.

1. The 5000 mAh battery in the real satellite is known to be over-dimensioned. Our aim was to find out how much. Hence, we performed a sequence of experiments, decreasing the size of the battery exponentially. The results (of the safe under-approximation) are displayed and explained in Figure 4. We found out that 1/4 of the capacity still provides sufficient guarantees (since the depletion risk calculated is in the order of 10^{-10}) to power the satellite for 1 year while 1/8 of the capacity, 625 mAh, does not. The following table compares the under- and over-approximations.

capacity (mAh)	**5000**	**2500**	**1250**	**625**
under-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$9.61 \cdot 10^{-96}$	$4.69 \cdot 10^{-43}$	$1.01 \cdot 10^{-15}$	0.00122
over-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$1.66 \cdot 10^{-63}$	$6.58 \cdot 10^{-31}$	$1.73 \cdot 10^{-10}$	0.03653

   The approximations thus compute the real probability of depletion up to very small absolute errors ranging from 10^{-63} for the 5000 mAh battery to 0.03531 for the 625 mAh battery.
2. We compared our results with a simple linear battery model of the same capacity.[3] This linear model is not uncommon in the satellite domain, it has for instance been used in the *Envisat* and *CryoSat* missions [18]. We obtain the following probabilities for battery depletion:

	linear battery model		**KiBaM**	
capacity (mAh)	5000	625	5000	625
under-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$1.76 \cdot 10^{-144}$	$8.53 \cdot 10^{-16}$	$9.61 \cdot 10^{-96}$	0.00122
over-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$1.86 \cdot 10^{-84}$	$2.94 \cdot 10^{-8}$	$1.7 \cdot 10^{-63}$	0.03653

[3] The linear model can be emulated using a KiBaM with diffusion rate $p \to \infty$. This has the effect that available and bound charge wells behave equally and thus deplete at the same time. To compute the numbers we used the same algorithm and discretisation constants δ, δ_g as for the corresponding KiBaM of the same size.

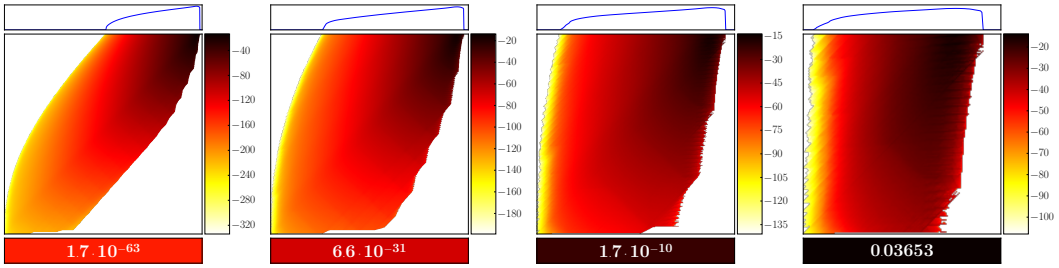


Figure 4 SoC under-approximation for different sizes of the satellite's battery after 1 year[a]. The leftmost SoC is with the original battery capacity, 5000 mAh. In each further plot, the battery capacity is halved, i.e. 2500 mAh, 1250 mAh, and 625 mAh. Note that all the densities are depicted on the logarithmic scale (ticks in the colorbar stand for the order of magnitude). We observe that only the smallest battery does not give sufficient guarantees. Its probability of depletion after 1 year is 0.0365; the probability decreases to $1.7 \cdot 10^{-10}$ already for the 1250 mAh battery. The smaller the battery, the more crucial is the distinction of available and bound charge as a larger area of the plots is filled with non-trivial density.

[a] Actually it is after 364 days, as this is in the middle of the charging phase. After 365 days the satellite is in eclipse and no density is exhibited along the upper limit.

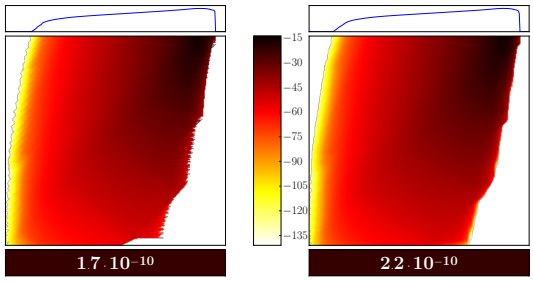


Figure 5 Load noise. SoC under-approximation of the 1 year run using the 1250 mAh battery with Dirac loads (left) and with noisy loads (right). We used Gaussian noise with standard deviation 5.

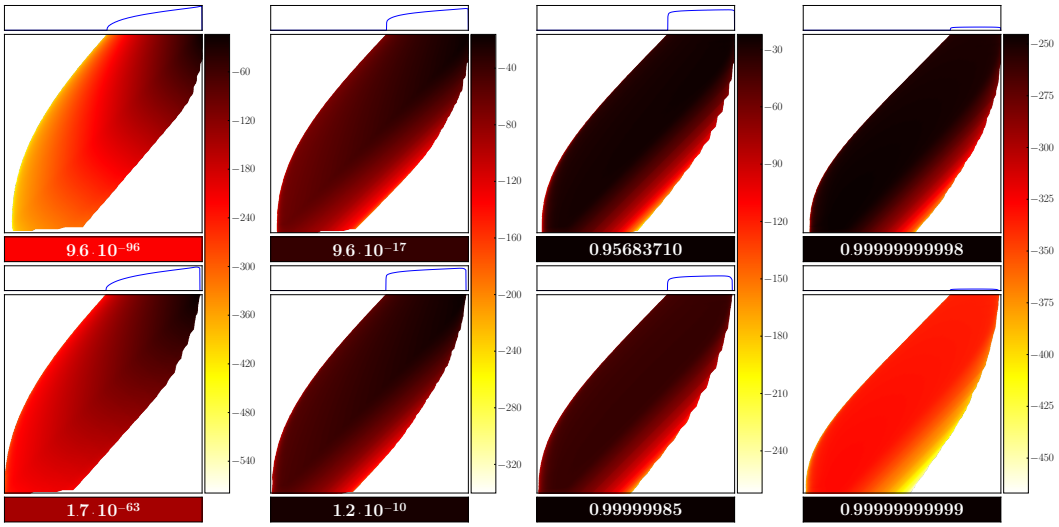


Figure 6 Number of solar panels. Top: The over-approximation of the full 5000 mAh battery with 9,8,7 and 6 solar panels. Bottom: The respective under-approximations on the same colorscale. Again, the ticks of the colorscale represent the order of magnitude of the densities.

The linear model turns out to be surprisingly (and likely unjustifiably) optimistic, especially for the 625 mAh battery.

3. We (computationally) simplified the two experiments above by assuming Dirac loads. To analyze the effect of the white noise, we compared the Dirac loads with the noisy loads, explained earlier, on the 625 mAh battery. As expected, the noise (a) smoothes out the distribution a little and (b) pushes a bit more of the distribution to full and empty states, see Figure 5.
4. Our reference satellite is a two-unit satellite, i.e. is built from two cubes, each 10 cm per side. In the current design, 9 of the 10 external sides are covered by solar panels, the remaining one is used for both radio antenna and camera. We thus conducted a robustness analysis with respect to solar infeed, by assuming that 1,2 and 3 solar panels break down. Figure 6 displays that the satellite can easily deal with 1 defective solar panel. If additional panels fail, the system runs out of energy rapidly with high probability.
5. The random KiBaM does not incorporate battery *aging*. In general, the degradation of a battery over time depends on many factors, most prominently how the battery was stored, which loads it was subjected to, how deeply it was discharged and at which temperatures it was used. We are not aware of a consensus method of how to model degradation of a Li-ion battery which is influenced by all of these factors. A measurable quantity related to battery age for our case study is the voltage drop when in eclipse. In-orbit measurements show that this voltage drop has worsened by 3% after one year of operation. For comparison purposes, we thus pessimistically assumed a battery with a capacity of only 4850 mAh (97% of 5000 mAh) from the beginning. Compared to the 5000 mAh battery the depletion probabilities are only slightly higher:

capacity (mAh)	**5000**	**4850**
under-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$9.61 \cdot 10^{-96}$	$1.73 \cdot 10^{-92}$
over-approximation of $\mathbf{Pr}[\mathit{depletion}]$	$1.66 \cdot 10^{-63}$	$5.58 \cdot 10^{-61}$

9 Alternative Approaches

The results reported above are obtained from a discretized abstraction of the stochastic process induced by the MTP and the battery, solved numerically and with high-precision arithmetic where needed.

One could instead consider estimating the probability z_t of the battery depletion using ordinary simulation techniques [19]. Considering a battery of capacity 5000 mAh, this would mean that about 10^{63} simulations traces are needed on average to observe the rare event of a depleted battery at least once. This seems prohibitive, also if resorting to massively parallel simulation, which may reduce the exponent by a small constant at most. A possible way out of this might lie in the use of rare event simulation techniques to speed up simulation [40].

The behaviour of KiBaM with capacity limits can be expressed as a relatively simple *hybrid automaton* model [21]. Similarly, the random KiBaM with capacity limits can be regarded as an instance of a *stochastic hybrid system* (SHS) [1, 3, 4, 8, 12, 37]. This observation opens some further evaluation avenues, since there are multiple tools available publicly for checking reachability properties of SHS. In particular, FAUST^2 [36], SISAT [17] and PROHVER [43, 16] appear adequate at first sight. However the random KiBaM system cannot be evaluated with FAUST^2, basically due to a model mismatch: The tool thus far assumes stochasticity in all dimensions, because it operates on stochastic kernels, while our model is non-stochastic in the bound charge dimension. The existing general theory about computing reach-avoid probabilities of so called partially degenerate stochastic processes [35] is not yet built into FAUST^2. The guarantees provided using these methods

are computed a priori on the basis of Lipschitz constants and do not scale well to the small absolute errors and large time horizons that are required for the satellite model. In our approach they are computed a posteriori (as the difference between under- and over-approximation). SiSat provides principal support for encoding all model aspects needed, yet the time horizon and precision needed seem unsurmountable [15]. Our ProHVer experiments failed for a similar reason, namely the sheer size of the problem. An extension of the stochastic network calculus to deal with energy [41] can in principle be employed to calculate depletion risks, by modelling the cumulative energy supply and energy demand as the arrival and service process of a queue, so as to capture the Fraction of Time Energy Not-Served (FTNS). Different from ours, that work assumes a linear battery behaviour and discretised time. Using a linear battery model causes underestimation of the depletion risk, as discussed in Section 8. All the above tools have not been optimized for dealing with very low probabilities as they appear in high dependability scenarios like the satellite case. The orders of magnitude difference between the smallest time step (5 minutes) and the time horizon (1 year) appear as another serious obstacle, but not for our approach.

10 Conclusion

Inspired by the needs of an earth-orbiting satellite mission, we extended in this paper the theory of kinetic battery models in two independent dimensions. First, we addressed battery charging up to full capacity. Second, we extended the theory of the KiBaM differential equations to a stochastic setting. We provided a symbolic solution for random initial SoC and a sequence of piecewise-constant random loads.

These sequences can be generated by a stochastic process representing an abstract and averaged behavioural model of a nano satellite operating in earth orbit, superposed with a deterministic representation of the solar infeed in orbit. We illustrated the approach by several experiments performed on the model, especially varying the size of the battery, but also the number of solar panels.

ESA is running a large educational program [2] for launching missions akin to GomX–1. The satellites are designed by student teams, have the form of standardized 1 unit cube with maximum mass of 1 kg, and target mission times of up to four years. The random KiBaM presented here is of obvious high relevance for any participating team. It can help quantify the risk of premature depletion for the various battery dimensions at hand, and thereby enable an optimal use of the available weight and space budget. Our experiments show that using the simpler linear battery model instead is far too optimistic in this respect.

For a fixed setup, one can also use the technology offered by us for optimal task scheduling: In the same way as we can follow a single SoC distribution, we can also branch into several distributions and determine which of them is best according to some metric. Taking inspiration from [42], this can be combined with statistical model checking so as to find the optimal task schedule of a given set of tasks.

Several extensions can and should be integrated in the model. Among them, temperature dependencies are of particular interest. A temperature change has namely opposing physical effects in solar panels and in the battery, having intriguing consequences such as piecewise exponential decay in the charging process. An extension that is particularly important for long lasting missions, is incorporating a model of battery wearout. So far we assume the battery capacity to be constant along the mission time. Notably, our contribution is the first to consider capacity limits in operation at all, as far as we are aware. As of now, our battery model is itself considered lossless, while in reality one never gets out as much energy as one has put in before. We are so far putting this phenomenon as a burden on the modeller side, namely to scale down the real charging current

to an effective charging current, that factors in the loss only while charging the battery. We are looking into ways to instead make these losses a genuine part of the KiBaM model.

Acknowledgements. The authors are grateful for inspiring discussions with Peter Bak, Morten Bisgaard, David Gerhardt and Jesper A. Larsen (GomSpace ApS), Erik R. Wognsen (Aalborg University), and other members of the SENSATION consortium, as well as with Pascal Gilles (ESA Centre for Earth Observation), Xavier Bossoreille (Deutsches Zentrum für Luft- und Raumfahrt) and Marc Bouissou (Électricité de France S.A., École Centrale Paris – LGI).

References

1 Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008. doi:10.1016/j.automatica.2008.03.027.

2 European Space Agency. ESA Cubesat program, October 2014. URL: http://www.esa.int/Education/CubeSats.

3 Eitan Altman and Vladimir Gaitsgory. Asymptotic optimization of a nonlinear hybrid system governed by a markov decision process. *SIAM Journal on Control and Optimization*, 35(6):2070–2085, 1997. doi:10.1137/S0363012995279985.

4 Henk A. P. Blom and John Lygeros, editors. *Stochastic Hybrid systems: Ttheory and Safety Critical Applications*, volume 337 of *Lecture Notes in Control and Information Science*. Springer Heidelberg, 2006. doi:10.1007/11587392.

5 Udi Boker, Thomas A. Henzinger, and Arjun Radhakrishna. Battery transition systems. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14, San Diego, CA, USA, January 20-21, 2014*, pages 595–606. ACM, 2014. doi:10.1145/2535838.2535875.

6 M. Brandl, H. Gall, M. Wenger, V. Lorentz, M. Giegerich, Federico Baronti, Gabriele Fantechi, Luca Fanucci, Roberto Roncella, Roberto Saletti, Sergio Saponara, Alexander Thaler, Martin Cifrain, and W. Prochazka. Batteries and battery management systems for electric vehicles. In Wolfgang Rosenstiel and Lothar Thiele, editors, *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 971–976. IEEE, 2012. doi:10.1109/DATE.2012.6176637.

7 Isidor Buchmann. *Batteries in a portable world*. Cadex Electronics Richmond, 2001.

8 Manuela L. Bujorianu, John Lygeros, and Marius C. Bujorianu. Bisimulation for general stochastic hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, pages 198–214. Springer, 2005. doi:10.1007/978-3-540-31954-2_13.

9 J. Cao, N. Schofield, and A. Emadi. Battery balancing methods: A comprehensive review. In *Vehicle Power and Propulsion Conference, 2008. VPPC'08. IEEE*, pages 1–6, Sept 2008. doi:10.1109/VPPC.2008.4677669.

10 Lucia Cloth, Marijn R. Jongerden, and Boudewijn R. Haverkort. Computing battery lifetime distributions. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh, UK, Proceedings*, pages 780–789. IEEE Computer Society, 2007. doi:10.1109/DSN.2007.26.

11 Robert M. Corless, Gaston H. Gonnet, D. E. G. Hare, David J. Jeffrey, and Donald E. Knuth. On the Lambert W function. *Adv. Comput. Math.*, 5(1):329–359, 1996. doi:10.1007/BF02124750.

12 Mark H. A. Davis. Piecewise-deterministic markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 353–388, 1984. URL: http://www.jstor.org/stable/2345677.

13 Marc Doyle, Thomas F. Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of The Electrochemical Society*, 140(6):1526–1533, 1993. doi:10.1149/1.2221597.

14 Maria Fox, Derek Long, and Daniele Magazzeni. Automatic construction of efficient multiple battery usage policies. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2620–2625. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-436.

15 Martin Fränzle. Personal communication. 2015.

16 Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*, pages 43–52. ACM, 2011. doi:10.1145/1967701.1967710.

17 Martin Fränzle, Holger Hermanns, and Tino Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*,

volume 4981 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2008. doi:10.1007/978-3-540-78929-1_13.

18 Pascal Gilles. Private communication. 2014.

19 Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976. doi:10.1016/0021-9991(76)90041-3.

20 GomSpace. Gomspace gomx-1, October 2014. URL: http://gomspace.com/?p=gomx1.

21 Thomas A. Henzinger. *Verification of Digital and Hybrid Systems*, chapter The Theory of Hybrid Automata, pages 265–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. doi:10.1007/978-3-642-59615-5_13.

22 Holger Hermanns, Jan Krcál, and Gilles Nies. Recharging probably keeps batteries alive. In Christian Berger and Mohammad Reza Mousavi, editors, *Cyber Physical Systems. Design, Modeling, and Evaluation – 5th International Workshop, CyPhy 2015, Amsterdam, The Netherlands, October 8, 2015, Proceedings*, volume 9361 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2015. doi:10.1007/978-3-319-25141-7_7.

23 Marijn R. Jongerden and Boudewijn R. Haverkort. Which battery model to use? *IET Software*, 3(6):445–457, 2009. doi:10.1049/iet-sen.2009.0001.

24 Marijn R. Jongerden, Boudewijn R. Haverkort, Henrik C. Bohnenkamp, and Joost-Pieter Katoen. Maximizing system lifetime by battery scheduling. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009, Estoril, Lisbon, Portugal, June 29 – July 2, 2009*, pages 63–72. IEEE Computer Society, 2009. doi:10.1109/DSN.2009.5270351.

25 Marijn Remco Jongerden. *Model-based energy analysis of battery powered systems*. PhD thesis, University of Twente, Enschede, December 2010. URL: http://doc.utwente.nl/75079/.

26 Edward A. Lee and David G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987. doi:10.1109/PROC.1987.13876.

27 Bor Yann Liaw, E. Peter Roth, Rudolph G. Jungst, Ganesan Nagasubramanian, Herbert L. Case, and Daniel H. Doughty. Correlation of arrhenius behaviors in power and capacity fades with cell impedance and heat generation in cylindrical lithium-ion cells. *Journal of power sources*, 119:874–886, 2003. doi:10.1016/S0378-7753(03)00196-4.

28 James F. Manwell and Jon G. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399–405, 1993. doi:10.1016/0038-092X(93)90060-2.

29 John Newman. Fortran programs for the simulation of electrochemical systems. URL: http://www.cchem.berkeley.edu/jsngrp/fortran.html.

30 Wilhelm Peukert. Über die Abhängigkeit der Kapazität von der Entladestromstärke bei Bleiakkumulatoren. *Elektrotechnische Zeitschrift*, 20:20–21, 1897.

31 Daler N. Rakhmatov and Sarma B. K. Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In Rolf Ernst, editor, *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001*, pages 488–493. IEEE Computer Society, 2001. doi:10.1109/ICCAD.2001.968687.

32 Venkatasailanathan Ramadesigan, Paul W. C. Northrop, Sumitava De, Shriram Santhanagopalan, Richard D. Braatz, and Venkat R. Subramanian. Modeling and simulation of lithium-ion batteries from a systems engineering perspective. *Journal of The Electrochemical Society*, 159(3):R31–R45, 2012. doi:10.1149/2.018203jes.

33 Venkat Rao, Gaurav Singhal, Anshul Kumar, and Nicolas Navet. Battery model for embedded systems. In *18th International Conference on VLSI Design (VLSI Design 2005), with the 4th International Conference on Embedded Systems Design, 3-7 January 2005, Kolkata, India*, pages 105–110. IEEE Computer Society, 2005. doi:10.1109/ICVD.2005.61.

34 Robin A. Sahner and Kishor S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Trans. Software Eng.*, 13(10):1105–1114, 1987. doi:10.1109/TSE.1987.232852.

35 Sadegh Esmaeil Zadeh Soudjani and Alessandro Abate. Probabilistic reach-avoid computation for partially degenerate stochastic processes. *IEEE Trans. Automat. Contr.*, 59(2):528–534, 2014. doi:10.1109/TAC.2013.2273300.

36 Sadegh Esmaeil Zadeh Soudjani, Caspar Gevaerts, and Alessandro Abate. Faust^2: Formal abstractions of uncountable-state stochastic processes. *CoRR*, abs/1403.3286, 2014. URL: http://arxiv.org/abs/1403.3286.

37 Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In Mathai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 6th International Symposium, FTRTFT 2000, Pune, India, September 20-22, 2000, Proceedings*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2000. doi:10.1007/3-540-45352-0_5.

38 Bart D. Theelen, Marc Geilen, Twan Basten, Jeroen Voeten, Stefan Valentin Gheorghita, and Sander Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *4th ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2006), 27-29 July 2006, Embassy Suites, Napa, California, USA*, pages 185–194. IEEE Computer Society, 2006. doi:10.1109/MEMCOD.2006.1695924.

39 Bart D. Theelen, Marc C. W. Geilen, Sander Stuijk, Stefan V. Gheorghita, Twan Basten, Jeroen P. M. Voeten, and Amir H. Ghamarian. Scenario-aware dataflow. Technical report, Eindhoven University of Technology, 2008. Technical Report ESR-2008-08. URL: http://www.es.ele.tue.nl/sadf/publications/TGSGBVG08.pdf.

40 Manuel Villén-Altamirano and José Villén-Altamirano. Restart: a straightforward method for fast simulation of rare events. In Deborah A. Sadowski, Andrew F. Seila, Mani S. Manivannan, and Jeffrey D. Tew, editors, *Proceedings of the*

26th conference on Winter simulation, WSC 1994, Lake Buena Vista, FL, USA, December 11-14, 1994, pages 282–289. ACM, 1994. doi:10.1109/WSC.1994.717150.

41 Kai Wang, Florin Ciucu, Chuang Lin, and Steven H. Low. A stochastic power network calculus for integrating renewable energy sources into the power grid. *IEEE Journal on Selected Areas in Communications*, 30(6):1037–1048, 2012. doi:10.1109/JSAC.2012.120703.

42 Erik Ramsgaard Wognsen, René Rydhof Hansen, and Kim Guldstrand Larsen. Battery-aware scheduling of mixed criticality systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications – 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II*, volume 8803 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2014. doi:10.1007/978-3-662-45231-8_15.

43 Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2010. doi:10.1007/978-3-642-14295-6_21.

Characterizing Data Dependence Constraints for Dynamic Reliability Using n-Queens Attack Domains*

Eric W. D. Rozier[1], Kristin Y. Rozier[2], and Ulya Bayram[3]

1 **Department of Computer Science, Iowa State University, Ames, IA, USA**
erozier@iastate.edu
2 **Department of Aerospace Engineering, Iowa State University, Ames, IA, USA**
kyrozier@iastate.edu
3 **Department of Electrical Engineering and Computing Systems, University of Cincinnati, Cincinnati, OH, USA**
http://orcid.org/0000-0002-8150-4053
bayramua@mail.uc.edu

Abstract

As data centers attempt to cope with the exponential growth of data, new techniques for intelligent, software-defined data centers (SDDC) are being developed to confront the scale and pace of changing resources and requirements. For cost-constrained environments, like those increasingly present in scientific research labs, SDDCs also may provide better reliability and performability with no additional hardware through the use of dynamic syndrome allocation. To do so, the middleware layers of SDDCs must be able to calculate and account for complex dependence relationships to determine an optimal data layout. This challenge is exacerbated by the growth of constraints on the dependence problem when available resources are both large (due to a higher number of syndromes that can be stored) and small (due to the lack of available space for syndrome allocation). We present a quantitative method for characterizing these challenges using an analysis of attack domains for high-dimension variants of the n-queens problem that enables performable solutions via the SMT solver Z3. We demonstrate correctness of our technique, and provide experimental evidence of its efficacy; our implementation is publicly available.

2012 ACM Subject Classification Embedded and cyber-physical systems, Data centers, Theorem proving and SAT solving
Keywords and Phrases SMT, data dependence, n-queens
Digital Object Identifier 10.4230/LITES-v004-i001-a005
Received 2016-01-11 **Accepted** 2016-12-09 **Published** 2017-02-28

Special Issue Editors Javier Campos, Martin Fränzle, and Boudewijn Haverkort
Special Issue Quantitative Evaluation of Systems

1 Notation

R	number of ranks (rows) of a Latin squares n-queens board; number of RAID groups in storage system under test
F	number of files (columns) of a Latin squares n-queens board; number of storage disks per RAID group

* An earlier version of this paper appeared in QEST'15 [4]. Some of the additional work consolidated here appeared in CFV'15 [27] and ISAIM'16 [28].

L	number of levels (height) of a Latin squares n-queens board; $L = R \cdot F$ with level l_i representing the problem associated with disk $r \cdot R + f$, or (r, f)
$x_{l,r,f} \in X$	index of a variable at level l, rank r, and file f of a Latin squares n-queens board X
X	a board; a set of $L \cdot R \cdot F$ variables each representing a the state of a single square, each with a single variable label
$X_C \subset X$	a set of variables labeling column C where $r = a$, and $f = b$, for some $a \in R$ and some $b \in F$
$Q = \{\Delta, \Lambda, \Gamma\}$	a finite set of symbols representing queens of three types
Δ	degenerate queens
Λ	linear queens
Γ	indirect queens
N	number of indirect queens on a board
A	a finite set of symbols representing squares under attack by each type of queen
ϵ	designator for an empty square
$V = Q \cup A \cup \epsilon$	set of variables that label squares
$S_{l,r,f}$	attack set (set of squares a queen attacks) such that $\forall s_i \in S_{l,r,f}, s_i = \lambda$ iff $x_{l,r,f} = \Lambda$, and $s_i = \gamma$ iff $x_{l,r,f} = \Gamma$
$D_{l,r,f} = S_{l,r,f} \cup x_{l,r,f}$	attack domain of a queen; set of squares a queen attacks or occupies
r_{s_i}	rank of square s_i
f_{s_i}	file of square s_i
ϕ	a disk file
$B_\phi = \{b_{\phi_0}, b_{\phi_1}, \ldots\}$	set of (typically fixed-size) blocks of file ϕ
$\mathcal{R}$	binary dependence relationship
$\mathcal{R}(\phi)$	dependence relation between blocks that are part of the same file such that for some $b_i, b_j, b_i\mathcal{R}(\phi)b_j$ if there exists some ϕ composed of B_ϕ where $b_i \in B_\phi$ and $b_j \in B_\phi$
σ	a reliability syndrome
$\mathcal{R}(\sigma)$	reliability syndrome dependence; there exists some σ such that $\sigma = b_i \oplus b_j \oplus \ldots$ then $b_i\mathcal{R}(\sigma)b_j$
P	set of $R \cdot F$ constants in the population constraint board designating available free-space per physical disk such that $\forall p_{r,f} \in P, p_{r,f} \in \mathbb{N}$
$p_{r,f} \in \mathbb{N}$	the population limit of column with rank r and file f
W	(constant) protection requirement for each level

2 Introduction

One of the largest challenges facing the storage industry is the continued exponential growth of Big Data. The growth of data in the modern world is exceeding the ability of designers and researchers to build appropriate platforms [33, 12] but presents a special challenge to scientific labs and non-profit organizations whose budgets have not grown (and often have been cut) as their data needs steeply rise. The NASA Center for Climate Simulation revealed that while their computing needs had increased 300 fold in the last ten years, storage needs had increased 2,000 fold, and called storage infrastructure one of the largest challenges facing climate scientists [10]. This trend has been driving reliance on commercial off the shelf (COTS) solutions to drive down the cost of data ownership. Despite its importance, the goal of affordable data curation comes at a cost in terms of reliability, creating a difficult-to-solve system-design-constraints problem.

To cope with the increase in cost, deduplication techniques are commonly used in many storage systems. Deduplication is a storage efficiency improvement technique that removes the duplicate substrings in a storage system and replaces them with references to the single location storing the

duplicate data. While this achieves a higher storage efficiency in terms of reducing the cost of ownership of a system, it can negatively impact the reliability of the underlying storage system since loss of a block with a high number of references means a critical number of files being lost unrecoverably [30].

Data reliability was previously improved using enterprise-class storage devices that typically suffer faults as much as two orders of magnitude less often than COTS storage devices. In the face of the exponential growth of the digital universe [36] the cost of this solution has become prohibitively expensive, inspiring a switch to near-line components, thus lessening storage reliability guarantees. While reliability could be improved through the addition of new hardware, today the scale of growth of inexpensive storage is being exceeded by the growth of Big Data.

In most storage systems reliability improvements are achieved through the allocation of additional disks in Redundant Arrays of Independent Disks (RAID) [25]. RAID arrays achieve reliability through the allocation of coding syndromes [26] that create dependence relationships in the storage system to allow recovery of files after failures. While RAID systems are incredibly effective at the task of improving reliability, they add to the cost of the storage systems in which they are deployed.

Methods used to increase reliability also increase the cost of maintaining the storage system, and the same is true for the methods that reduce the cost; they also reduce reliability. In order to meet these cost and reliability constraints, and find a way to break the proportional relationship in between, previously we conducted a study where we have documented that systems are often over-provisioned, and this over-provisioning level is highly predictable using intelligent systems algorithms [31]. Using these models, we proposed that dynamically allocated reliability syndromes could be created and stored in this excess capacity to improve reliability without the addition of new hardware [3]. Based on this result, it is now possible to modify traditional RAID schemes to dynamically allocate new syndromes for reliability in over-provisioned space through the risk-averse prediction of available storage over the next epoch of operation of a storage system. Furthermore this can be done while maintaining quality of service (QoS) and availability of the storage system, while simultaneously providing maximum additional reliability. The only assumption is that the additional syndromes can be placed in a way that respects data dependence constraints. The ability to predict the expected level of over-provisioning allows us to create software-defined data centers that can allocate virtual disks made up of free space compiled from across the data center to hold additional reliability syndromes. An unsolved challenge that stands in the way of this technique, however, is the development of algorithms that account for complex data dependencies such as existing reliability syndromes and deduplication, providing a strategy for syndrome storage and new RAID relationships in a performable way that maximizes the additional number of reliability syndromes that can be allocated without violating the dependence constraints on those syndromes.

2.1 n-Queens

In order to solve these dependence constraints, we cast our problem into a unique variant of the n-queens problem. We chose n-queens for several reasons. First and foremost, when fully constructed, our board resembles the classic 3-dimensional Latin board configuration [21, 16], and we recognized that the independence requirements for new reliability syndromes could be represented as a metaphor of the squares in this Latin board that represent legal captures. To place another syndrome into such a square would violate independence, and as n-queens concerns itself with a placement of new queens (syndromes) on a board (disk array) such that none attack each other (independence is preserved) the formulation seemed a natural choice. We map a RAID array into a mathematical representation of a chess board with a set number of *ranks* (defining the

y-axis) and *files* (defining the x-axis). We propose a quantitative solution for virtual disk allocation in software-defined data centers, respecting all dependence constraints within the data center, or, when no such configuration exists, identifying the unsatisfiability of the problem. This method allows us to take advantage of the over-provisioned space without constraining our problem to traditional RAID geometries. We propose solving this problem quantitatively by mapping it to an innovative variation of the n-queens problem that utilizes a 3D Latin board configuration [21, 16], nontraditional queen types and attack domains, and population limits on the number of indirect queens placed within certain bounds. While this formulation differs from traditional n-queens in several ways, we make the argument that it is a difference in *degree*, and not in *kind*. Utilizing n-queens lets us not only utilize common and well-explored metaphors, but it also allows us to leverage generalized solvers and packages built for n-queens, and allows others to modify our solution to fit their needs should other attack patterns be required to represent dependence relations we do not concern ourselves with, such as meta-data relationships. By formulating our problem as a variant of n-queens our solver can be used in other domains, or by other variants of the disk layout problem we are solving simply by modification of the attack domains exhibited by the queens in our problem.

The challenge of defining dynamic syndromes is inherently characterized by a well-defined set of constraints: total number of disks, current disk utilization, distribution of unutilized space, existing dependence relationships due to RAID reliability syndromes, and deduplication relationships. By creating a mapping to n-queens under these constraints, we can intuitively represent the problem in a way that facilitates validation and harness the power of the Satisfiability Modulo Theories (SMT) solver Z3 to return a constraint-satisfying solution, or determine that a solution cannot exist. Z3 [8] is a very efficient and freely-available solver for SMT, which is a decision problem for logical first-order formulas with respect to combinations of background theories including the uninterpreted functions integral to our solution. The n-queens problem is a classic way to represent such a constraint satisfaction problem [22, 32] and a common benchmark for such a solver [17]. Classification as a constraint satisfaction problem that can be solved by Z3 has proven to be successful in other design domains, such as automating design of encryption and signature schemes [1].

2.2 Previous Work

In our previous work [4] we formulated a variant of the n-queens problem using a basic set of constraints, and showed informally and empirically that our method could sometimes generate satisfying solutions. We empirically characterized the difficulty of finding a solution in terms of the number of queens, and the population coverage ratio. We also demonstrated that deduplication has the general effect of making the problem less likely to be satisfiable. We extended this work in [28] by formally casting our constraints into subsets requiring global and local scoping with respect to changing level protection requirements. This, in essence, gave us the ability to utilize partial solutions to the global constraints problem to reduce the total time necessary to evaluate cascading solutions to progressively harder variants of a given problem. Since finding a satisfying disk layout for some level of protection W may not be possible, but one for $W - i$ for some i may exist, this allowed us to adopt a solution method of solving easier variants, and using them to speed up the solution of progressively better-protected systems within a finite time bound. We additionally showed, empirically, that when we examine large samples of random disk layouts, that satisfiability of the problem is probabilistic, and has a regular structure examined as a function of available disk space, and the entropy of that space.

2.3 Application to Embedded Systems

A particular challenge for the realm of embedded systems is the performance requirement induced by the fact that embedded systems are often subject to both real-time constraints, and additionally that they lack considerable processing power. One of the primary targets of our technologies are for systems for which extremely high reliability, beyond that demanded by consumer systems, is required. These systems are often difficult, if not impossible, to repair and include embedded storage systems for satellites, remote probes and rovers used by NASA, and planned spacecraft (both manned and unmanned) for long mission profiles like the Autonomy Architecture Habitat. In addition, it would be advantageous for our technology to run on small embedded systems, even when part of a larger more capable storage system, so that it can be implemented as a plug-and-play technology that sits between user- and system-level requests, and the storage architecture translating file requests, when needed to account for additional reads and/or writes to enable the higher level of reliability transparently.

Given these goals, one of the primary focuses of this paper is the derivation of a system that is not only correct, but that can be employed with acceptable overhead, and cascading solutions allowing for real-time constraints to be accounted for. The ideal system would be one in which successively harder problems are solved one after another (or in parallel if possible) until the deadline is reached. At that point the best solution computed to date is used for system layout. We present such a system here.

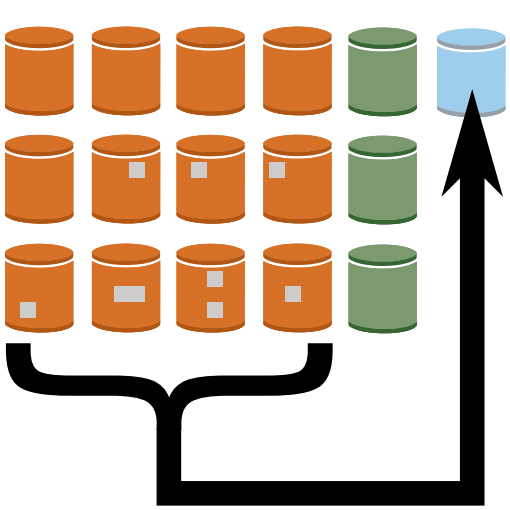
2.4 Novel Contributions

Our contributions in this paper include a new quantitative solution for the problem of dynamic allocation of new reliability syndromes while respecting dependence constraints to improve the reliability of software-defined data centers without the addition of new hardware. We extend our previous work by giving a formal definition of our problem with accompanying proofs of correctness for a mapping of this problem to a variation of the classic n-queens problem, thus enabling efficient analysis via powerful SMT solvers like Z3. We provide an implementation in Z3 for python and include a case study demonstrating the effectiveness of our technique. This new solution will serve as the core for a dynamic allocation system to be used in software-defined data centers that will be deployed at the laboratories of partner organizations.

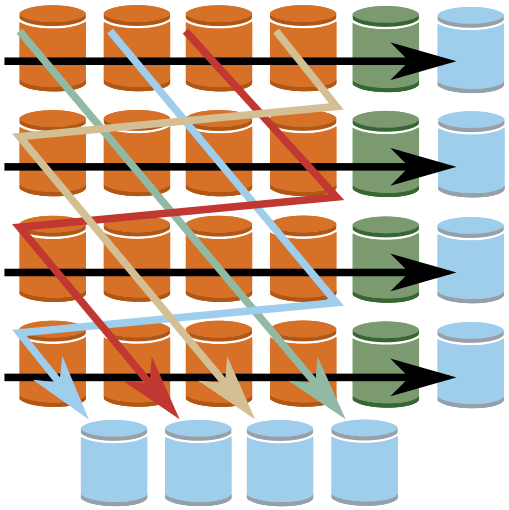
This paper is organized as follows: Section 3 provides background on dependence relationships in storage systems, and related work in novel RAID geometries. Section 4 introduces an encoding for this problem in a variant of n-queens, mapping the problem of data layout strategies that respect all data dependence constraints while maximizing additional syndrome coverage for any given dataset, to the problem of placing novel queen types on a Latin chess board. We formalize these definitions and the resulting constraints in Section 5, and give formal mappings to Z3 for implementation in Section 6. We provide experimental results demonstrating the efficacy and efficiency of our approach in Section 7. Finally, Section 8 concludes and points to future work.

3 Characterizing File System Dependence

As we have shown in our previous work [31, 3], it is possible to predict the future storage resource needs of the users in a system. In recent work [3], we have modeled user behaviors using the training data we obtained from a real system to create and train Markov models, and predicted the future disk usage needs of the users in an on-line fashion, and compared the results with the test data we also obtained from the same system to measure the prediction performance. We have observed that with a good clustering method and fine parameter tuning, it is possible



(a) Example of a virtual disk being constructed out of overprovisioned space.



(b) Example of independent syndrome calculation as the XOR parity of diagonals.

Figure 1 Example allocations of virtual disks from over-provisioned space.

Table 2 Annual rates of block loss (ABL) per system type with varying numbers of additional syndromes (n_{synd}) allocated.

RAID5 configuration	ABL (no syndromes)	ABL ($n_{synd} = 2$)	ABL ($n_{synd} = 3$)
5+1	$1.79\text{x}10^{5}$	$1.31\text{x}10^{-7}$	$1.92\text{x}10^{-15}$
8+1	$4.60\text{x}10^{5}$	$1.02\text{x}10^{-6}$	$5.06\text{x}10^{-14}$
10+1	$8.06\text{x}10^{5}$	$2.76\text{x}10^{-6}$	$1.79\text{x}10^{-13}$

to predict user behaviors and resource requirements. We have used this method for predicting over-provisioning, and allowing for dynamic improvement of reliability through the allocation of additional syndromes by creating new *virtual disks* using any over-provisioned storage that are found to be independent of the current RAID grouping as shown in Figure 1a. Our experiments on real storage system data have shown that even when being incredibly risk adverse, we can allocate between three and four additional syndromes more than 50% of the time, and on average allocate two additional syndromes for all of the data, and three additional syndromes for more than 90% of the data, dramatically improving the reliability of the system [3]. We analyzed these improvements on systems with one petabyte of primary storage with initial RAID5 configurations of 5+1, 8+1, and 10+1 over which we introduce two and three additional syndromes after predictions. Changes in reliability are measured using the rate of annual block loss (ABL), when taking into account whole disk failures and latent sector errors. Table 2 illustrates the calculated ABLs for three RAID5-configured primary storage systems, each provisioned for a maximum capacity of one petabyte. The steep increase in the reliability represented by decreases in ABL rates as the number of allocated syndromes increases shows the promise of such predictive analysis and dynamic allocation.

3.1 Reliability Syndromes

The typical way of addressing reliability concerns in large-scale storage systems has been through the generation of syndromes that can be used to detect faults, prevent those faults from manifesting

as failures, and repair those failures when new resources are available. The most basic type of syndrome that can be allocated is that of XOR parity [6]. Consider a set of disks that contain an array of blocks, the atomic unit of reading and writing in a file system. We can treat each of these blocks as a vector of bytes and generate a syndrome by performing some calculation on each byte in the vector. In order to tolerate the loss of a single disk in some set of n disks we need to compute a syndrome P, which allows for the recovery of any lost block. One of the simplest methods for doing so is XOR parity:

$$P = D_0 \oplus D_1 \oplus D_2 \oplus \ldots \oplus D_{n-1}\,.$$

We can then write P to a new disk, independent from those containing blocks used in its computation, creating an array of $n+1$ disks. The loss of any one disk, including the one containing P, will not result in the loss of data. If some disk D_j fails and cannot be read or written normally, we can perform equivalent operations on the remaining disks to account for this *degraded* state. A read to D_j can be performed by reading the working $n-1$ disks and the disk containing P and generating D_j from the result as

$$D_j = D_0 \oplus D_1 \oplus D_2 \oplus \ldots \oplus D_{j-1} \oplus D_{j+1} \oplus \ldots \oplus D_{n-1} \oplus P$$

(where $2 < j < n-1$ for this example, but without loss of generality for other cases). When new hardware is acquired (or allocated from hot spares available in the storage system), the entire disk containing all blocks associated with the failed disk that contained D_j can be recovered in a similar manner.

In order to tolerate the loss of any two disks two independent syndromes must be calculated, here referred to as P and Q. Without providing additional disks, in order to construct a new independent syndrome we utilize the algebra of a Galois field $\mathbf{GF}(2^8)$ [2]. The representation of this algebra is cyclic utilizing group or ring theory. We utilize elements g called generators of the Galois field such that g^n doesn't repeat until it has exhausted all elements of the field except $\{00\}$, where any numeral in $\{\}$ is a hexadecimally-represented Galois field element. We defer a full discussion of Galois field algebra to the literature [14]. For n disks where $n \leq 255$ we compute:

$$P = D_0 \oplus D_1 \oplus D_2 \oplus \ldots \oplus D_{n-1}\,, \tag{1}$$

$$Q = g^0 \cdot D_0 \oplus g^1 \cdot D_1 \oplus g^2 \cdot D_2 \oplus \ldots \oplus g^{n-1} \cdot D_{n-1}\,. \tag{2}$$

The loss of a single data drive can be recovered using the normal XOR parity method described previously. The loss of P or Q can be recovered simply by recomputing using the above formulas. The loss of any single data drive, and the loss of Q can be recovered by first recovering the data drive using XOR parity, and then recomputing Q. Recovering P, or the loss of two data drives is somewhat more involved, and the discussion of the method is left to the literature [2].

3.2 Allocation of New Syndromes

Allocation of new syndromes in order to increase the reliability through the deployment of RAID5 XOR parity syndromes [25] or RAID6 Galois-field based syndromes [2, 7] becomes somewhat trickier due to the requirement of independence. Additional syndromes can, in theory, be allocated using techniques such as erasure coding, which would generate still new independent syndromes. These methods, however, generally have a severe impact on performance, and as a result, lower the QoS of the system [20]. As such, we focus on alternative RAID geometries to make use of additional XOR parity and Galois-field based syndromes. To do so we must overcome the problem of our requirement of independence.

In this paper, we propose an efficient method for allocation of additional syndromes. Additional coverage can be provided using non-traditional RAID geometries as shown in Figure 1b. While the idea of using non-traditional RAID geometries itself is not new, and has been explored in previous studies [34, 24, 23], prior work in this field has always maintained the assumption that the layout of the RAID arrays is pre-defined. Instead, we propose the creation of dynamic per-stripe geometries using over-provisioned space in an existing data center.

When creating non-traditional RAID geometries, care must be taken to respect data dependence relationships [29] to ensure that the new RAID strategy improves reliability. We consider two types of data dependence relationships, one resulting from pre-existing RAID groups, and the other from data deduplication [30].

A typical method for reliability syndrome generation is XOR parity. In a situation such as that shown in Fig. 1a, data may be made more reliable by creating a new dependence between currently independent data. So given blocks a, b, c, and d stored on separate physical hardware, a new block $z = a \oplus b \oplus c \oplus d$ can ensure that if any block is lost, for example c, it can be easily recreated as $c = a \oplus b \oplus d \oplus z$, adding to the reliability of the underlying file system [25]. Reliability can be further extended through the use of Galois fields [6], and in theory with erasure codes [9], however codes patent encumbrance has effectively removed performable erasure code algorithms from use [13]. In practice this means for any set of initially dependent data, reliability can be increased (given sufficient space) via creation of two independent syndromes.

Additional reliability syndromes can be allocated using additional blocks not already linked through a syndrome-related dependence, such as a, f, k, and p in Fig. 1. The difficulty inherent in allocating these new syndromes is ensuring independent sets of blocks can be identified, along with independent free space in the storage system. As the storage system becomes fuller over time, the difficulty of this problem increases exponentially, necessitating efficient solution techniques.

We consider three types of data dependence relationships in our analysis. The first are *file dependence* relationships. We consider data in our file systems to be divided into blocks (typically of fixed size) with each file ϕ being composed of a set of blocks $B_\phi = \{b_{\phi_0}, b_{\phi_1}, \ldots\}$. Blocks that are part of the same file have a file dependence relation represented by $\mathcal{R}(\phi)$ such that for some b_i, b_j, $b_i \mathcal{R}(\phi) b_j$ if there exists some ϕ composed of B_ϕ where $b_i \in B_\phi$ and $b_j \in B_\phi$. Secondly, we consider *reliability dependence*. Such a dependence, represented by $\mathcal{R}(s)$, exists between blocks b_i, b_j if both b_i and b_j participate in reliability syndrome s. Thus if there exists some s such that $s = b_i \oplus b_j \oplus \ldots$ then $b_i \mathcal{R}(s) b_j$. Lastly, we consider *deduplication dependence* relationships. These relationships are much like those found in file dependence relationships, and can be defined in the same way, differing only in that for a deduplicated block b_i, it can participate with multiple files in dependence relationships, so $b_i \in B_k$ does not preclude that some B_l also exists such that $b_i \in B_l$, $l \neq k$. For convenience, we will also use the notation $\mathcal{R}$ without a subscript to indicate the presence of any dependence relationship, regardless of the type. These relationships become important when defining a new syndrome s' to protect some block b_p. When defining s' as a set of blocks $S' = \{b_p, b_0, b_1, \ldots\}$ such that $s' = b_p \oplus b_0 \oplus b_1 \oplus \ldots$ it is important to pick blocks such that for $b_i \in S' \setminus b_p, b_i \mathcal{R} b_p$ is false; otherwise the new syndrome will not provide the expected improvements to reliability as independence is a fundamental assumption for syndrome construction.

4 n-Queens with Dynamic Domains of Attack

In order to solve our problem and find a data layout that allows us to build virtual disks that are independent of the data they are protecting, we provide a mapping of our problem into a variant on the classical n-queens [35] constraint satisfaction problem with few alterations.

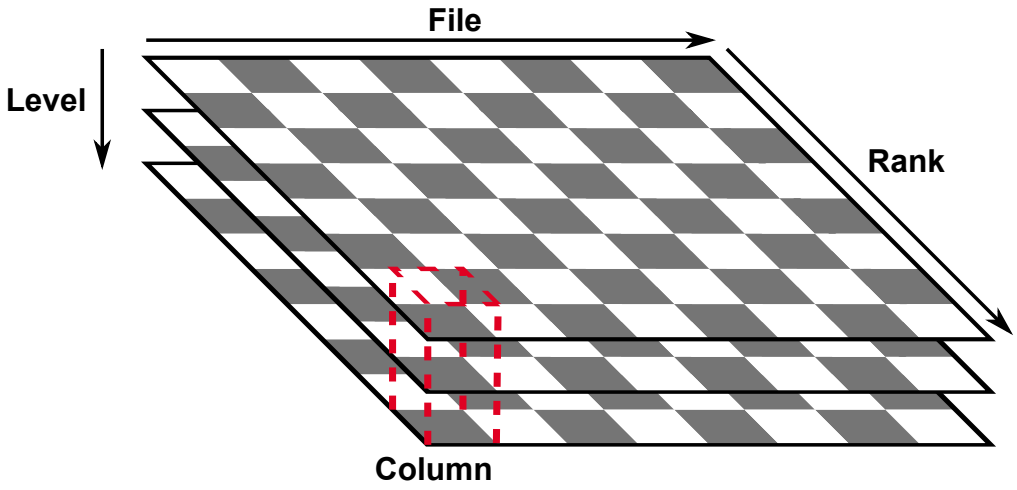


Figure 2 Example Space with dimensions 3x8x8.

First, we adopt a Latin board, allowing us to examine our problem in a three-dimensional space [21, 16]. We define this space according to three axes, the level, rank, and file, as shown in Figure 2. We further define a column on this Latin board as the set of squares defined by a fixed rank and file across all levels of the board. Each column in our Latin board corresponds to a disk within our data center, with each rank consisting of a traditional RAID group. Levels represent independent sub-problems solving for data independence for each disk in turn. Thus, in practice, given a problem with R ranks and F files, we construct our board with $L = R \cdot F$ levels.

We represent the state of dependence relationships in a file system by placing queens on our boards, using their attack domains to represent file dependence relationships. For any level l on our board, this level is used to solve a sub-problem for the lth disk in our data center (numbered in rank-major order, such that if the disk is in rank r and file f, the level that solves its independence constraints is $l = r * F + f$). The full attack domain of all queens on level l represents those disks on which the lth disk depends. We call this lth disk for level l the principle disk for that level. To represent these dependence relationships, however, we specify the attack domain definitions for each queen to match the dependence relationships we must represent. We introduce three new queen types each with a unique attack domain.

- **Degenerate Queens** – a degenerate queen is so-named because it attacks only a single square, that which it is occupying. Degenerate queens are used to represent the disk being protected, and disks containing deduplicated blocks upon which files on that disk depend. Degenerate queens are used to exclude a square on a level from the solution space of new dynamic RAID groupings. The attack domain of a degenerate queen is illustrated in Figure 3.
- **Linear Queens** – a linear queen's attack domain is defined to include both its own square and $F - 2$ squares on the board extending in a line from the queen, potentially wrapping around the board as if it were a toroidal-board as discussed originally in the class of modular n-queens problems [11]. Linear queens can be used to represent existing RAID groups, or new dynamic RAID groups with more traditional geometries. Two example attack domains for linear queens are illustrated in Figure 4.[1]

[1] While we allow linear queens to attack in any direction as a matter of completeness of our variant n-queens definition, we note that our method only makes use of linear queens that attack along ranks towards squares in higher-numbered files, wrapping toroidally.

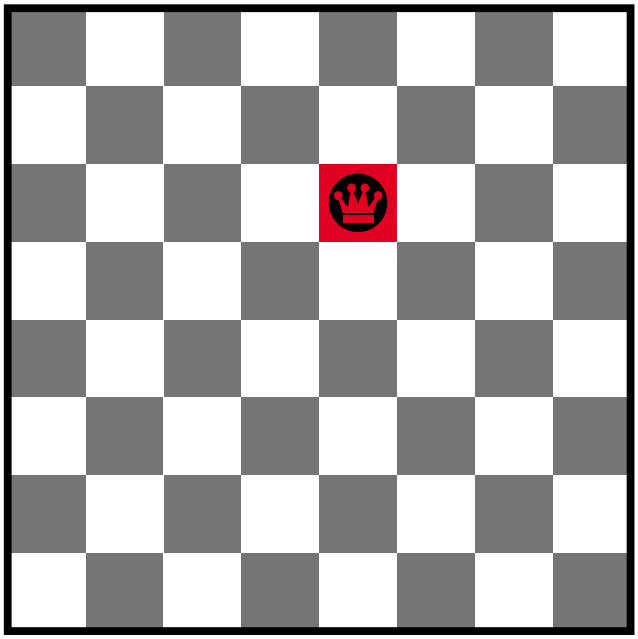


Figure 3 Example attack domain of a single degenerate queen.

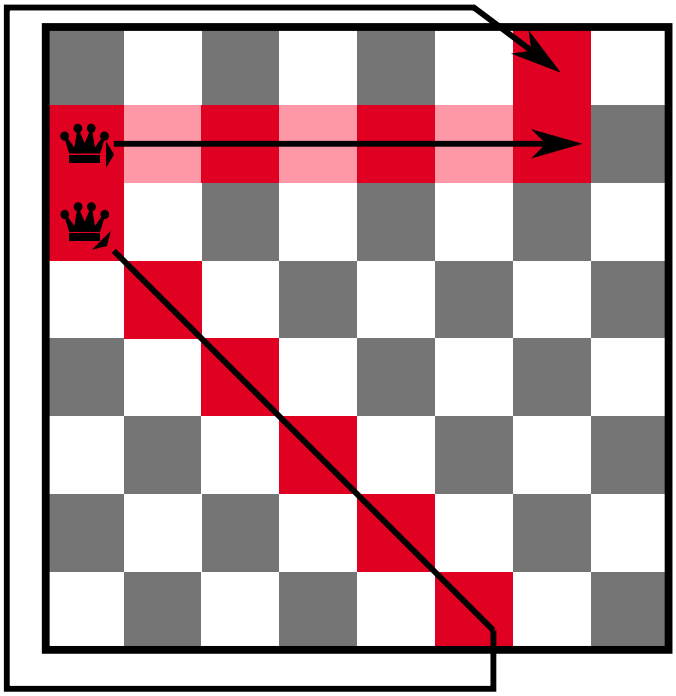


Figure 4 Example attack domains of two linear queens.

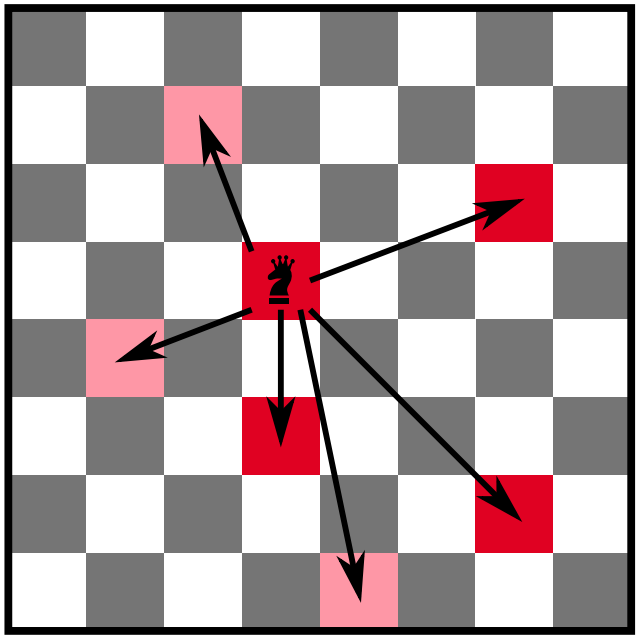


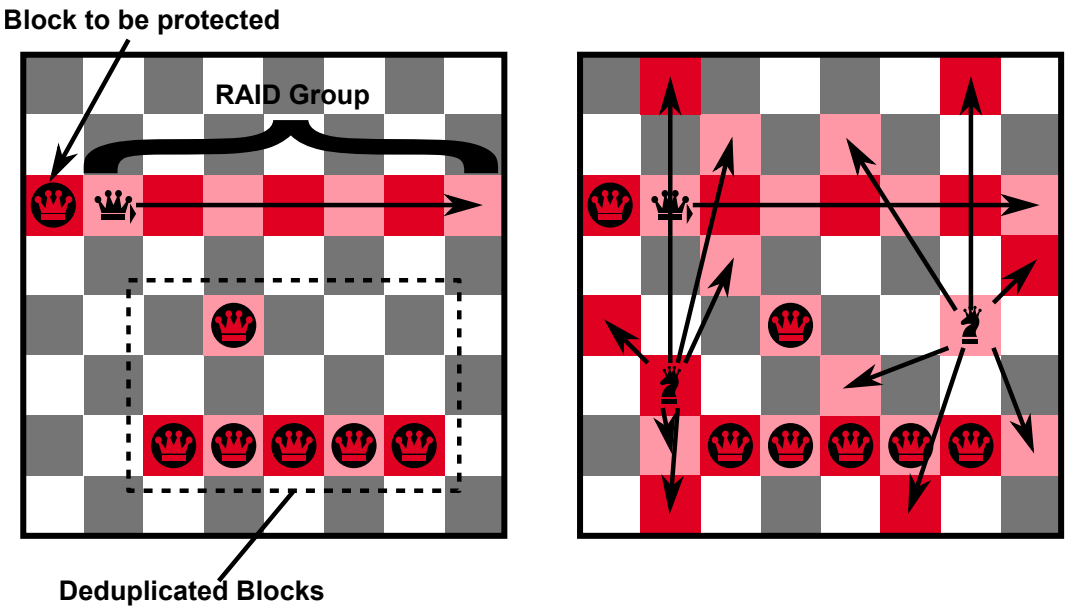
Figure 5 Example attack domain of a single indirect queen.

- **Indirect Queens** – the final type of queen we introduce is an indirect queen, whose attack domain consists of its own square, and $F-2$ other squares on the board, each within a rank unique to the queen's attack domain. The attack domain of an indirect queen is illustrated in Figure 5. An indirect queen can attack with almost any imaginable pattern, so long as it attacks $F-2$ squares and those squares are on unique ranks. This allows for the formulation of arbitrary RAID geometries that still respect dependence relationships arising from standard RAID protections. The indirect queen itself is used by our problem to represent the disk on which a new syndrome will be stored, and the $F-2$ squares in its attack domain represents those other disks participating in the syndrome calculation.

In order to solve the problem of independent syndrome placement, and the creation of new dynamic RAID groupings, we begin with a pre-defined board, based on the state of the data center, that contains a number of degenerate and linear queens representing this system state, such as the example shown in Figure 6a. We then proceed to place new indirect queens on the board with each indirect queen representing the storage location of a new pair of XOR and Galois field parity syndromes, and the attack domain of that queen representing the independent disks to use to form a new dynamic RAID group associated with those syndromes.

5 Formal Problem Representation

In this section we provide a formal representation of our problem accompanied by some helpful proofs, define our representation, and provide constraints for use in SMT solving that allow the production of strategies for reliability improvement, if any such strategy exists. Our solution is intuitive and easy to validate as the n-queens problem is a classic way to represent such a constraint satisfaction problem [22, 32]; n-queens variations are common benchmarks for SMT solvers [17], so it is easy to choose a good solver. We represent our problem in the domain of a Latin-



(a) Example of initial constraints when protecting a block on disk 0 of RAID group 2 that has references to deduplicated blocks on six other disks. Degenerate queens are used to include the disks containing the initial and deduplicated blocks in the attack domain, and a linear queen is used to include the RAID group in the attack domain

(b) An example solution with two additional syndromes. Indirect queens occupy the spaces corresponding to the disks where the new syndromes will be stored; their attack domains include all disks protected by the new syndrome.

Figure 6 Representation of a single level of an 8x8x64 board.

squares [21, 16] variant of the n-queens problem, using multiple levels of the Latin-squaresboard to represent separate, yet dependent, subproblems, using novel variant queen types with unique attack domains, and population constraints. In our representation, the board represents the physical disk media with each column in the Latin-squares board representing a separate physical disk.

▶ **Definition 1** (Square)**.** A square represents a discrete part of the n-queens problem that has a state that can either represent its occupancy by a queen (including the type of the queen), that it is part of the attack domain of a queen (i.e., some queen could capture a piece were it on that square), or that it is empty. This state is encoded for any given square as a variable drawn from a finite set $V = Q \cup A \cup \{\epsilon\}$ where Q is a finite set of symbols representing queens of three types, A is a finite set of symbols representing squares under attack by each type of queen, and ϵ is an empty square.

We allow each square to contain only one queen, or be part of an attack domain as part of the implicit requirement of n-queens where a valid placement results in no queen attacking another.

▶ **Definition 2** (Board)**.** For ease of representation, we utilize a three-dimensional matrix: a Latin-squares variant of n-queens with the dimensions L levels, R ranks, and F files, as shown in Fig. 2. A board is a set of $L \cdot R \cdot F$ variables indexed $x_{l,r,f} \in X$. Each variable represents the state of a square and has an assignment from a finite set $V = Q \cup A \cup \{\epsilon\}$ that represents its state, where Q is a finite set of symbols representing queens of three types, A is a finite set of symbols representing squares under attack by each type of queen, and ϵ is an empty square.

We use the term *column* to indicate a set of variables $X_C \subset X$ where $r = a$, and $f = b$, for some $a \in R$ and some $b \in F$.

Given an array of $R \cdot F$ disks arranged into traditional RAID groupings of F disks per group, each level of the board represents a separate constraint satisfaction problem for a separate set of data associated with a given physical disk. Specifically, $L = R \cdot F$ with level l_i representing the problem associated with disk $r \cdot R + f$, or (r, f). For each set of data on a given physical disk we construct the initial data dependencies by assigning queens and their attack domains to the variables representing a given level.

Queens, and the squares they attack, are used to represent dependence relationships between data on the physical disks represented by the board. We characterize the squares that a queen is said to attack as the *attack domain* of the queen, and the combination of squares that a queen occupies and attacks as the *attack set*.

▶ **Definition 3** (Queen)**.** A queen is a symbol from the set $Q = \{\Delta, \Lambda, \Gamma\}$, that can be assigned to any free variable. The three queen symbols differ in their allowed attack domains and are called degenerate queens (Δ), linear queens (Λ), and indirect queens (Γ).

Initial conditions for our data dependence constraints are constructed using two special types of queens, *degenerate queens* and *linear queens* that differ from the standard queens of the classical problem in terms of their attack domains.

▶ **Definition 4** (Attack Domain)**.** The attack domain of a queen at position $x_{l,r,f}$ is defined by its position, and a set of additional squares called it's *attack set* given by the set $S_{l,r,f}$. This attack domain, $D_{l,r,f} = S_{l,r,f} \cup \{x_{l,r,f}\}$, represents every square a queen attacks or occupies.

▶ **Definition 5** (Attack Set)**.** An attack set is a set of variables $S_{l,r,f}$ assigned labels from the set $\{\lambda, \gamma\}$ to designate they are attacked by a queen such that $\forall s_i \in S_{l,r,f}, s_i = \lambda$ iff $x_{l,r,f} = \Lambda$, and $s_i = \gamma$ iff $x_{l,r,f} = \Gamma$. Note that there is no attack set associated with a degenerate queen ($x_{l,r,f} = \Delta$) because the attack domain of a degenerate queen contains only the square containing the degenerate queen itself.

▶ **Definition 6** (Degenerate Queen)**.** A degenerate queen is represented by Δ and has no corresponding attack symbol in A. This is because the attack domain of a degenerate queen contains only the square containing the degenerate queen itself, i.e. $D_{l,r,f} = \emptyset \cup \{x_{l,r,f}\}$.

▶ **Definition 7** (Linear Queen)**.** A linear queen is represented by Λ and has the corresponding attack symbol λ. The size[2] of a linear queen's attack set is always equal to $N - 2$ and must satisfy Constraint 1.

▶ **Constraint 1** (Linear Queen Attack Set)**.** The attack set of a linear queen assigned to variable $x_{l,r,f}$ must be such that the size[3] of a linear queen's attack set is always equal to $F - 2$ and either Constraint 1.1, 1.2, or 1.3 is satisfied. All elements of a linear queen's attack set must reside on the same level.

▶ **Constraint 1.1** (Constant File)**.** The attack set S of a linear queen at $x_{l,r,f}$ satisfies the *Constant File* constraint iff $\forall s_i \in S$ the file of s_i is equal to f.

[2] We assume that any additional protection provided uses precisely the same RAID configuration as disks in default RAID groupings. This is assumed both for simplicity and performance reasons, but can be relaxed without loss of generality.

[3] Both for simplicity and performance reasons, we assume that any additional protection provided uses the same RAID configuration as the default RAID groupings. This assumption can be relaxed without loss of generality.

▶ **Constraint 1.2** (Constant Rank). The attack set S of a linear queen at $x_{l,r,f}$ satisfies the *Constant Rank* constraint iff $\forall s_i \in S$ the rank of s_i is equal to r.

▶ **Constraint 1.3** (Unique Rank and File). The attack set S of a linear queen at $x_{l,r,f}$ satisfies the *Unique Rank and File* constraint iff $\forall s_i, s_j \in S$ the file of $s_i(f_{s_i})$ and $s_j(f_{s_j})$ are such that $f_{s_i} \neq f, f_{s_j} \neq f$, and $f_{s_i} \neq f_{s_j}$, and the rank of $s_i(r_{s_i})$ and $s_j(r_{s_j})$ are such that $r_{s_i} \neq r, r_{s_j} \neq r$, $r_{s_i} \neq r_{s_j}$, and $s_i \neq s_j$.

▶ **Definition 8** (Initial Board). An initial board is a set of initial conditions for a file system coded as a set of fixed values for a subset of X. These values represent the initial system and consist of a placement of degenerate queens from Definition 6 and linear queens from Definition 7.

We then try and find a solution that satisfies all of our constraints, and that allows us to place W or more *indirect queens* on our board, where each indirect queen represents a new syndrome to be allocated for reliability, and its attack domain represents the new data dependencies associated with that syndrome.

▶ **Definition 9** (Indirect Queen). An indirect queen is represented by Γ and has the corresponding attack symbol γ. The size[4] of an indirect queen's attack set is always equal to $F - 2$ and must satisfy Constraint 2.

▶ **Constraint 2** (Indirect Queen Attack Set). The attack set S of an indirect queen assigned to variable $x_{l,r,f}$ must be such that $\forall s_i, s_j \in S$ the rank of $s_i(r_{s_i})$ and $s_j(r_{s_j})$ are such that $r_{s_i} \neq r$, $r_{s_j} \neq r$, $r_{s_i} \neq r_{s_j}$, and $s_i \neq s_j$. All elements of an indirect queen's attack set must reside on the same level.

Each indirect queen is able to provide both XOR parity, and Galois field parity for its attack domain, provided the disk has available space. This space constraint holds for an entire column as well, as each column represents a single physical disk. This necessitates the representation of column-wise population constraints in the form of a *population constraint board*.

▶ **Definition 10** (Population Constraint Board). In addition to the defined set of $L \cdot R \cdot F$ variables that make up the board, we add a set P of $R \cdot F$ such that $\forall p_{r,f} \in P, p_{r,f} \in \mathbb{N}$. We call this set of constants the population constraint board.

The population constraint board tracks the available free-space per physical device, and constrains the total placement of indirect queens within a column.

▶ **Constraint 3** (Column Indirect Queen Population Limit). Each column may be assigned a population limit $p_{r,f} \in \mathbb{N}$. The total number of all indirect queens within that column must not exceed this limit. We generate $R \cdot F$ new constraints for each combination of unique r and f such that $r \in [0, R-1]$ and $f \in [0, F-1]$

$$\left(\sum_{l \in [0,(R \cdot F)-1]} (x_{l,r,f} = \Gamma) \right) \leq p_{r,f} \, .$$

We further constrain our problem from the traditional variants of n-queens by requiring not only that no queen placed on the board attack another queen, but also by requiring that no two

[4] Both for simplicity and performance reasons, we assume that any additional protection provided uses the same RAID configuration as the default RAID groupings. This assumption can be relaxed without loss of generality.

queens attack the same square. This requirement that attack domains not intersect is necessary to ensure the independence of calculated syndromes. Recall that each level of our board represents the protection problem for a given block. Informally, if two queens were both to attack the same square, it would mean the syndromes they represent are both calculated from a block on the same disk. Those two syndromes would then no longer be independent, as if the disk failed from that they were both calculated, they would suffer a correlated failure. We represent this with Constraint 4.

▶ **Constraint 4** (Non-intersection of Attack Domains)**.** In addition to the constraint that no queen falls within the attack domain of another queen, we further constrain the problem by specifying that no attack domain may intersect the attack domain of another queen.

We add a level protection requirement as a constraint to specify that all disks are protected by at least P additional syndromes per block on the disk that contains data.

▶ **Constraint 5** (Level Protection Requirement)**.** For a given level l, the sum of the number of all indirect queens on that level must be greater than or equal to the protection requirement W. We generate L new constraints for each $l \in [0, (F * R) - 1]$ of the form

$$\sum_{r \in [0,R-1], f \in [0,F-1]} (x_{l,r,f} = \Gamma) \geq W\,.$$

This protection requirement W is level-independent and applies to all levels of a board, i.e. all blocks are required to have the same number of additional syndromes allocated.[5]

▶ **Definition 11** (Satisfying Assignment of Variables in X)**.** We define a satisfying assignment to be an assignment of each variable in X to exactly one value from $V = Q \cup A \cup \epsilon$ such that this assignment satisfies Constraints 1, 2, 3, 4, and 5.

▶ **Theorem 12.** *Constraint 4 holds for any solution: the attack domains of any two queens never intersect.*

Proof. The proof follows from our construction and problem representation. From Definition 2, a board is defined by a set of variables, X, where each variable has a single assignment from $V = Q \cup A \cup \{\epsilon\}$, the set of variable assignments representing queens, attack domains, or empty squares not under attack. From Definition 11, all variables must have exactly one assignment from V, thus two queens of different types may not both attack the same square as doing so would require that variable to have a non-unique assignment and instead take on the value of the tuple, $\{\lambda, \gamma\}$. Thus the only case where the attack domains of two queens might overlap is when those queens are of the same type. From Definitions 6, 7, and 9 and Constraints 1 and 2 we know that a valid assignment for a board must contain $F - 2$ squares in the attack set of any queen (except a degenerate one) [4]. So if there are N non-degenerate queens of a given type on a board there must be $N(F - 2)$ squares assigned to their attack domains. If two queens of the same type had overlapping attack domains, fewer than $N(F-2)$ squares would be assigned to the attack domains of those queens, violating Definitions 7 and 9, as well as violating Constraints 1 and 2. ◀

▶ **Theorem 13.** *The set of Constraints 1, 2, 3, 4, and 5 are both necessary and sufficient to ensure that any satisfying assignment to X represents a potential layout for a set of new independent reliability syndromes. If no such satisfying assignment is found, no such layout exists.*

[5] This level-independence can be relaxes, but is not recommended as it opens the question of block importance, for which there currently exists no domain-inspecific metric, and no metric at all for some domains.

Proof. For a satisfying assignment to represent a data layout that improves the reliability of the underlying data storage system it *must* provide:

Condition 1. A new, and empty, block that may be used to store the new independent reliability syndromes.

Condition 2. A set of blocks that can be used to calculate the new independent reliability syndromes.

Condition 3. W such sets per block to establish the required additional level of protection.

Constraint 3 is sufficient for Condition 1, as each indirect queen itself represents the storage location of a new syndrome and the population board is created by identifying empty blocks in the storage system by Definition 10. Constraint 5 is sufficient for Condition 3, by definition. These are both trivially sufficient for their respective conditions as well, by definition.

Condition 2, that of independence, relies on a given block being used once, and only once, for each independent form of syndrome calculation. Thus the same block may be involved in both a Galois field operation [2] and XOR parity calculation [6] but may not appear twice for in the equation for a given block. Two types of parity calculations are relevant for our proof. The first are those in the pre-existing storage system. The second are those needed for newly computed syndromes. From Definition 8 we know that the initial board consists of all pre-existing syndromes represented by linear queens from Definition 7. Constraint 1 requires that for a given queen it's attack domain must take on the form of a straight line having either constant rank but independent file from Constraint 1.2, constant file but independent rank from Constraint 1.1, or independent rank and independent file from Constraint 1.3 ensuring independence. Newly computed syndromes are represented by indirect queens, given by Definition 9, placed by the SMT solver. These indirect queens have their attack sets constrained by Constraint 2 that also ensures independence. Taken together Constraints 1 and 2 ensure any syndrome computed or pre-existing is independent of every other block in its computation. Constraint 4 ensures any syndrome computed or pre-existing is independent of every other syndrome computation used for the same block. Thus we prove overall sufficiency as Constraints 1, 2, 3, 4, and 5 are sufficient for Conditions 1, 2, and 3.

If Constraint 3 is violated, not enough free space is available and Condition 1 does not hold. If Constraints 1, 2, or 4 are violated, independence does not hold, and Condition 2 is violated. If Constraint 5 is violated not all blocks are protected by the requisite number of syndromes, and Condition 3 is violated. Thus Constraints 1, 2, 3, 4, and 5 are necessary. ◀

5.1 Improving Tractability Through Variable Domain Reduction

In order to improve tractability of our solution we attempt to reduce the possible solution space by reducing the cardinality of the domain of variables in X given by $V = Q \cup A \cup \{\epsilon\}$. We propose that our problem representation can be simplified without loss of generality through the collapse of the variable domain.

▶ **Definition 14** (Variable Domain Collapse)**.** We define our variable domain collapse as a process by that our domain V is collapsed from $V = \{\{\Delta, \Lambda, \Gamma\} \cup \{\lambda, \gamma\} \cup \epsilon\}$ to $V = \{\Delta, \Gamma, \gamma, \epsilon\}$ via the following reduction semantics:

$$x \in X | x = \Lambda \quad \rightarrow \quad x = \Gamma \tag{3}$$

$$x \in X | x = \lambda \quad \rightarrow \quad x = \gamma \tag{4}$$

▶ **Theorem 15.** *Let X' be a satisfying assignment of X where each element $x \in X$ is assigned exactly one value from $V = Q \cup A \cup \epsilon$ such that this assignment satisfies Constraints 1, 2, 3, 4,*

and 5. For each x in X such that $x = \Lambda$ we assign $x = \Gamma$. For each x in X such that $x = \lambda$ we assign $x = \gamma$. The resulting new assignment X'' is one where each element $x \in X$ is assigned exactly one value from $V = \{\Delta, \Gamma, \gamma, \epsilon\}$ such that this assignment satisfies Constraints 1, 2, 3, 4, and 5. Given that X' is a satisfying assignment, X'' is also a satsifying assignment.

Proof. Application of the reduction semantics given by Definition 14 results in treating as equivalent the pairs of symbols (Λ, Γ) and (λ, γ). This has the potential to alter the correctness of Theorem 12 that relies on Definition 11, to prove all variables must have exactly one assignment from V to prove two queens of different types may not both attack the same square. If such were the case Theorem 12 shows doing so would require that square to have a non-unique assignment and instead take on the value of the tuple, $\{\lambda, \gamma\}$. Definition 14, however, no longer requires a variable to take on a non-unique assignment, as λ and γ are now treated as equivalent. We can prove the result is still correct by noting Constraints 1 and 2 both require attack sets of $F - 2$ squares for each unique queen. As such the attack domains of linear and indirect queens must not intersect or either Constraint 1 or Constraint 2 would be violated resulting in a solution that does not meet the requirements set out by Definition 11. Thus any solution that is satisfying without Definition 14 is also satisfying under Definition 14 as any satisfying placement of a linear queen is also a satisfying placement of an indirect queen. While the converse is not true, linear queens are placed only as part of the initial board given by Definition 8. So long as this initial placement satisfies 1 under the equivalence given by Definition 14, then any satisfying solution under Definition 11 is still a satisfying assignment. ◀

▶ **Definition 16** (Reduction relations on variable assignments)**.** We define a reduction relation over variable assignments.

$$x_{l,r,f} \to \Delta \quad \text{if} \quad x_{l,r,f} \in A = \{\lambda, \Delta, \Lambda\}\,.$$

Definition 16 allows us to reduce the total set of possible values a variable can be assigned by recognizing that the importance of linear and degenerate queens, and their attack domains, can be reduced to a single value representing a square that a new indirect queen, or it's attack domain, cannot occupy due to Constraint 4

5.2 Computational Complexity

While the less restrictive attack domains of our three new queen types would seem to make the problem less difficult than traditional n-queens, and more equivalent to the trivial n-Rooks problem [5, 37], the population constraints board serves to complicate the problem of queen placement, especially as the number of levels we must solve for grows polynomially. The population constraints board has the effect of creating attack domains in the z-axis when enough queens are placed in a column. Figure 7 shows the relative difficulty of solving this new variant n-queens problem vs. traditional n-queens, and highlights the additional complexity despite the more easily satisfied attack domains of our variant queens. While this graph suggests that scalability is an issue, we will address scalability concerns in Section 7 through a proposed compositional approach.

6 Solving with Z3

In order to determine if a given data center state and desired protection level is satisfiable, we utilized Z3 and encoded our problem in the form of variables and uninterpreted functions forming an SMT problem.

We encode these constraints into Z3 using the assertions shown in Figure 8. Assertion 5 sets the domain of the variables representing the board and ensures satisfaction of Constraint 4.

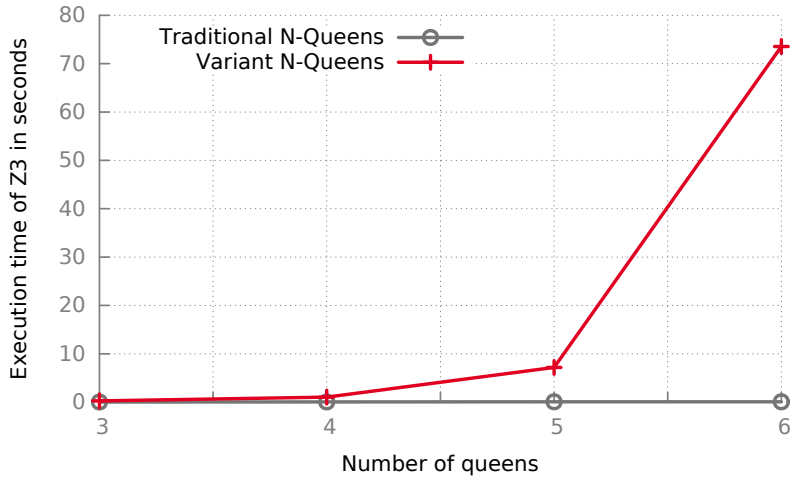


Figure 7 Comparison of solution times for Z3 given the placement of Y queens on a $Y \times Y$ traditional n-queens board and a $Y^2 \times Y \times Y$ variant n-queens board from our problem.

$$\forall l \in L, \forall r \in R, \forall f \in F(x_{l,r,f} \in \{\lambda, \Delta, \Lambda, \epsilon\}) \tag{5}$$

$$\forall l \in L, \forall r \in R, \forall f \in F((r = \lfloor l/F \rfloor) \rightarrow (x_{l,r,f} = \Delta)) \tag{6}$$

$$\forall l \in L, \forall r \in R, \forall f \in F((\exists b | b \mathcal{R} l) \rightarrow (x[l, r, f] = \Delta)) \tag{7}$$

$$\forall r \in R, \forall f \in F(p_{r,f} \geq \sum_{l} (x_{l,r,f} = \Gamma)) \tag{8}$$

$$\forall l \in L(\sum_{\forall r, \forall f} (x_{l,r,f} = \lambda) \geq W \cdot |F|) \tag{9}$$

$$\forall l \in L(\sum_{\forall r, \forall f} (x_{l,r,f} = \Lambda) \geq W) \tag{10}$$

Figure 8 Equations which characterize the n-queens constraints for our variant problem as Z3 assertions.

Assertion 6 removes the entire rank containing the block we are protecting on a given level from the solution space of indirect queen placement due to preexisting dependence relationships, and ensures that Constraint 1 is satisfied. Assertion 7 removes any disk containing a block deduplicated with a block on the disk we are trying to protect due to a preexisting dependence relationship. Population limits are maintained by assertion 8. Assertion 9 satisfies a weaker form of Constraint 2 when coupled with Constraint 3 as it allows for an indirect queen to potentially have a larger than necessary attack set. Since this relaxation of Constraint 2 would result in a more reliable system, we utilize it to make the problem easier for Z3. Finally Assertion 10 ensures satisfaction of Constraint 5.

6.1 Cascading Solver

We utilized a cascading approach to our solver as discussed in our previous work [28]. This cascading solver addresses the possibilities of real-time constraints by first generating a further constrained model based on the Z3 assertions that are global with respect to the choice of level protection requirements, i.e. Assertions 5, 6, 7, and 8. Our cascading solver takes advantage of Z3's capability to manage constraints in the form of a stack containing assertions. This stack of assertions may contain nested scopes that can be created and destroyed. We examine a set

of pre-computed tables that characterize the likelihood of a satisfying solution for our problem for various protection levels as a function of the Population Constraint Board. Specifically we examine both the number of resources available, and the entropy of those available resources. The entropy of resource allocation can be interpreted as a diversity metric. High entropy systems are those with more uniform resource distribution. Low entropy systems tend to have resources concentrated on a few disks.

We then solve for the global scope using Z3's `SimpleSolver` to establish the initial model of our problem, and use this initial model for our cascading solution for the subsequent problems for $q \in [q_p, q_s]$. The problem corresponding to q_s is known to be satisfiable due to the classification of our current system based on its resources and entropy, and we can prove that some q_s exists experimentally due to the fact that for $q = 1$ all possible systems are satisfiable. This solution for q_s guarantees that we will find some (possibly non-optimal) solution, providing additional reliability. We then solve the problems for the remaining $q \in [q_p, q_s - 1]$ in ascending order (corresponding to higher probability of satisfiability) until we find an unsatisfiable result.

It is important to note that this solution technique does not result in the addition of a single additional syndrome at a time, as this would result in a non-optimal solution, and potentially lead us to believe no satisfying solution exists, when in fact one does due to accidentally overconstraining our problem through false assumptions. Instead it a successive solution of harder problems based on nested scoping in Z3, attempting to utilize the available time before our deadline is reached to find **a** solution, and then improve on that solution if time remains.

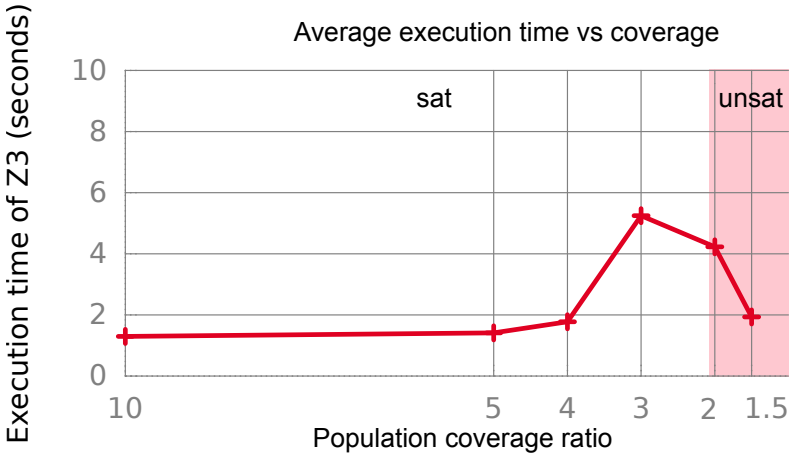
7 Experimental Results and Validation

In order to validate our results we conducted experiments with random initial system states for both population constraints boards, and data deduplication constraints. All experiments were run using a single EC2 c4.large instance with 2 virtual CPUs and 3.75 GiB of RAM. We implemented our solver to print out the resulting boards in a human readable format and hand checked the results, also collecting performance statistics for the Z3 solutions. The simulated systems were characterized by their total storage capacity in terms of terabytes with the assumption that each system consisted of a set of 1TB disks in an 8+2 configuration. Thus a 160TB system would consist of 160 disks in 16 ranks of 10 files each.

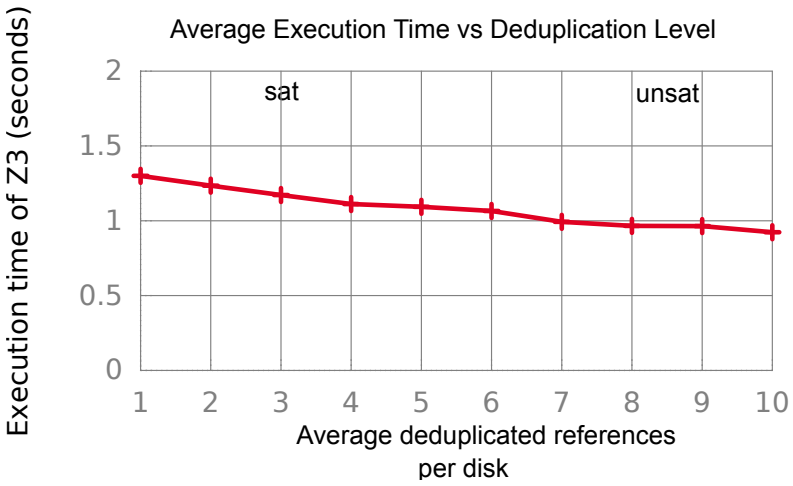
Figure 9 along with Figure 7 provide a summary of the results of our experiments. We found a sharp satisfiability cliff accompanying the population constraints board that correspond to the probability of a rank having no available space. This suggests an important observation to account for when moving forward with a full implementation of software-defined data centers, namely that balancing of over-provisioned space can be critical when such space becomes rare and the data center approaches capacity if the excess space is to be used to improve reliability. This limit is approached even more swiftly for large systems in which many levels are competing for the same population constraints within a rank.

We found the problem to be less sensitive to deduplication. While we eventually found a region of unsatisfiable problems at higher deduplication ratios, the more random placement of deduplicated references ameliorated their constraints on the solution space. It should also be noted that such constraints only became an issue at very high levels of deduplication, suggesting that deduplication based dependences are not as difficult to account for as might be expected.

The exponential growth in runtimes is somewhat concerning, as it seems to limit this solution technique to smaller storage systems, which presents a problem when confronted with the exponential growth of Big Data. Large-scale systems could potentially take infeasible amounts of time to solve if solved directly. As a consequence of this result we propose that larger systems be solved



(a) Run time and satisfiability of random problems as the population constraints board is made more restrictive.



(b) Run time and satisfiability of random problems as the deduplication ratio is increased.

Figure 9 Partial summary of experimental results.

compositionally. For instance, while a 160TB system takes 74 seconds to solve, if the system is blocked into two 80TB systems by decomposing individual ranks a satisfying solution for each system can be found within 2.5 seconds each, and can be solved in parallel. The exponential improvements found through compositional solution, coupled with the embarrassingly parallel nature of the SMT sub-problems created by partitioning the system by rank provides a very scalable alternative to attacking the entire problem at once. This method has the advantage of respecting dependence relationships, as when decomposed into separate sub-problems all relationships can be accounted for between sub-models in a trivial fashion since their proposed solutions will include only those ranks within a given sub-problem.

Since the population constraint board is known as part of the system state, we can choose to sort each rank into one of S subproblems based on the rank of the population constraint board associated with the rank of the Latin board. The satisfiability of the subproblems, depending primarily on these population constraints, can be maximized by sorting the ranks on the basis of the population constraints associated with their columns. Using such a solution we are able to

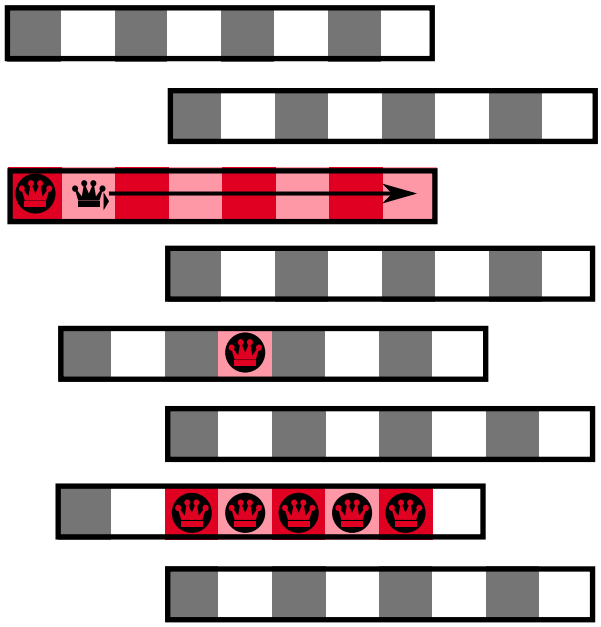


Figure 10 Example of row-wise decomposition.

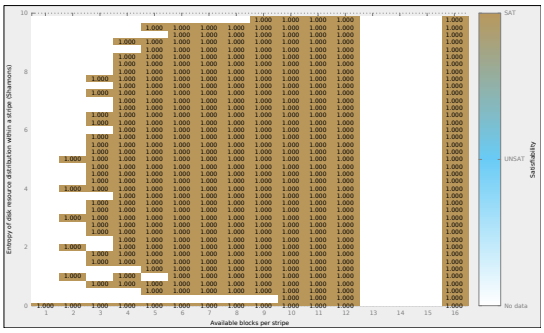
scale linearly with the size of our data center. We note the potential to further improve solution by partitioning the initial system on a row-wise basis, as shown in Figure 10. By sorting rows based on available resources for additional syndromes, and the distribution of those resources, we may be able to optimize our compositional solution.

In Figure 11 we show the results of experiments we conducted for various board configurations of varying resource levels, and the entropy of those resources, giving the proportion of observed boards that were satisfiable, or unsatisfiable. We observe clear trends with respect to available resources (more resources indicates a higher probability of satisfiability), and the Shannon entropy of the distributions of those resources (higher entropy distributions are more satisfiable). Thus, for a given board, we can characterize the satisfiability of the sub-boards resulting from a row-wise decomposition, allowing us to optimize the sub-boards generated to maximize the probability the resulting system will be satisfiable for a chosen W.

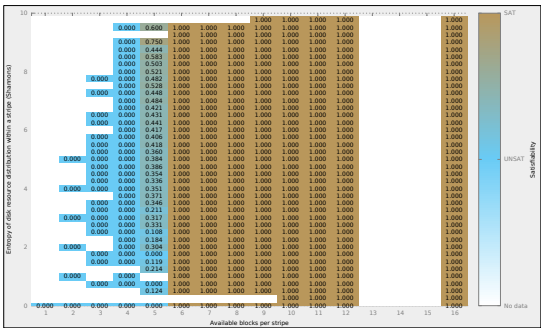
The run times of the resulting boards were likewise highly regular with respect to resources and their entropy, as shown in Figure 12. Again we note faster run times for higher numbers of available resources, and higher entropy distributions of those resources.

7.1 Application to Embedded and Resource Constrained Systems

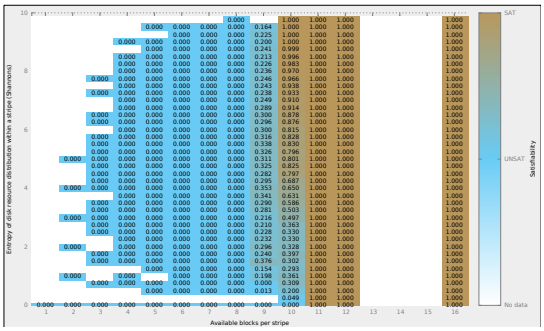
The results of the experiments shown in Figures 11 and 12 describe a highly regular space can be generated when our problem is characterized in terms of available resources and entropy. This space is learnable for problems of a given size via estimation through a random sampling of the feature space to ensure coverage of various resource allocation levels, and entropies. We utilized the Kumaraswamy distribution [19] due to it's beta-like properties for creating distributions with varying skewness and kurtosis, and due to the closed-form nature of it's inverted distribution function [15] to generate a thorough exploration of the space of possible distributions for disk resources within a pre-existing RAID stripe. By controlling skewness and kurtosis we were able to produce a number of different resource entropies easily, to ensure even coverage of the underlying space.



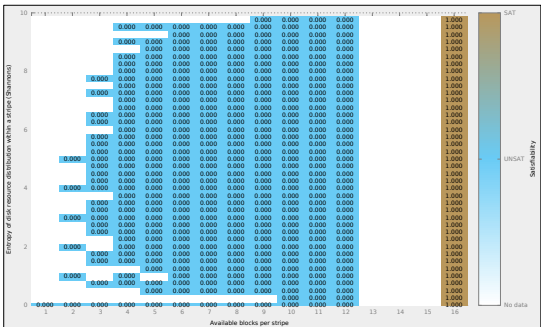
(a) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 1$.



(b) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 2$.



(c) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 3$.

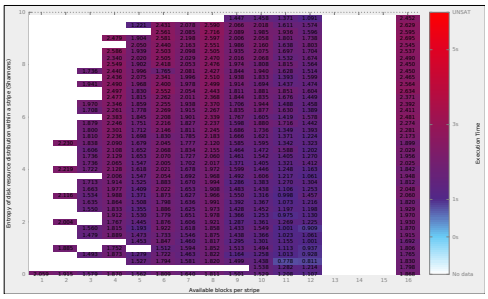


(d) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 4$.

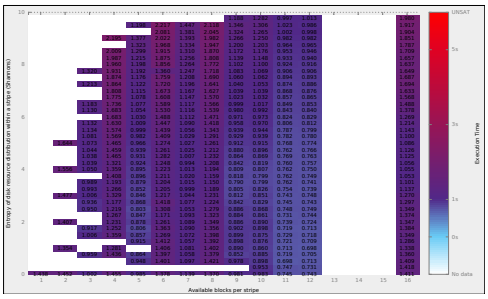
Figure 11 Probability of a satisfiable solution for $W \in [1, 4]$ based on available resources, and the entropy of resource distribution.

We make the observation from Figure 11 that a given resource level and entropy level of those resources can be used to characterize the underlying system as either **unsatisfiable** for a given protection level W, **satisfiable** for a given protection level W, or **probabilistically satisfiable** for a given protection level W. In the case that our problem is probabilistically satisfiable, we note that the probability of satisfiability generally increases with increased entropy of resource distribution. We explain this finding intuitively by observing that higher diversity of positions (corresponding with a higher entropy) more often results in the possibility of a satisfiable solution as it eliminates competition between queens for independent resources within a given rank.

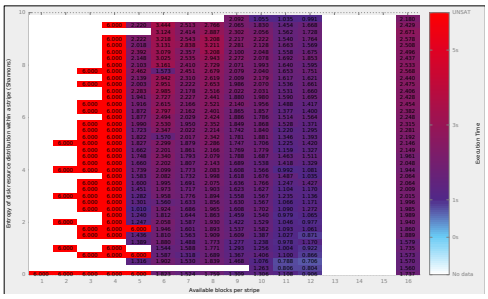
By generating this table in advance we can apply our technique selectively by first choosing some protection level that our tables indicate is solvable for a system with the currently observed resource levels and resource entropy, and solving that problem first. If time remains before our deadline is reached we can these use Figure 12 in conjunction with Figure 11 to estimate if it is reasonable to attempt a new, higher protection, solution. In this way even though not all disk layouts are satisfiable for all protection levels we can maximize the likelihood of finding a satisfying solution, and the probability of achieving a problem of high protection levels.



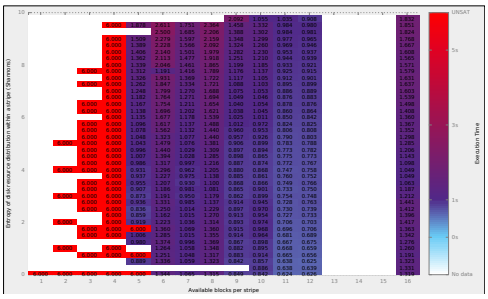
(a) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 1$.



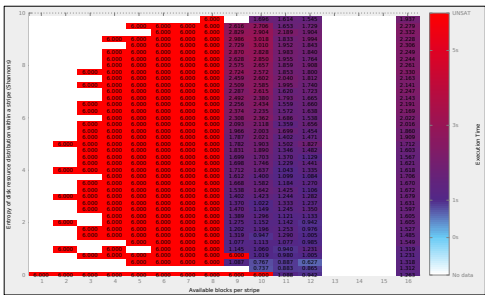
(b) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 1$.



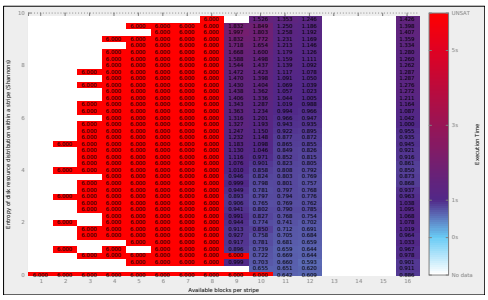
(c) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 2$.



(d) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 2$

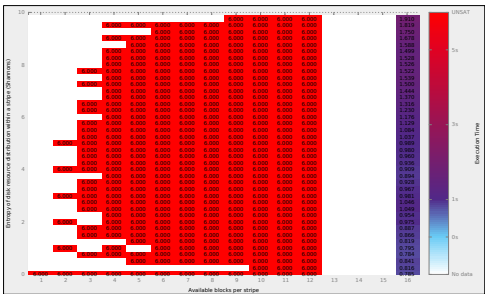


(e) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 3$.



(f) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 3$

(g) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 4$.



(h) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 4$

Figure 12 Maximum and mean time to solution for $W \in [1, 4]$ based on available resources, and the entropy of resource distribution.

8 Conclusions

In this paper we have presented a novel formulation of the n-queens problem using a modular Latin board, non-traditional queen variants, and column-based population constraints. This formulation serves as a translation of data dependence constraints and the problem of virtual syndrome creation for software-defined data structures into SMT allowing for efficient solution that allows for improved reliability with no additional hardware in over-provisioned systems. While our problem grows exponentially more difficult for larger storage systems, we provide a scalable way to achieve similar levels of protection through rank-wise decomposition of the problem space using population-constraint sorting into embarrassingly parallel subproblems.

The overhead of this method is minimized by several factors, the first being the ability of the cascading solution to learn the feasibility of the solution space to avoid searching for solutions to protection levels for which the likelihood of finding a satisfying solution is low, and second due to the low probability of the need to rebuild from these more complex syndromes. Our system can function as if it is protected only by RAID 5 or RAID 6 protections, ignoring the extra allocated blocks according to the schemes discussed in [31, 3]. This new method will form the basis for a performable dynamic RAID allocation system for use in large-scale storage systems serving cost-constrained organizations, providing an intelligent software stack that will help to combat the exponential growth of Big Data.

8.1 Future Work

Now that we have an efficient, scalable method for determining whether there exists a dynamic reliability syndrome that satisfies its data dependence constraints, we can move onto looking at other interesting optimizations. Currently we either generate a single strategy for additional syndrome allocation, or prove that no such allocation exists. However, the option is now open for us to harness more of the power of Z3 to query the solution space to optimize for secondary considerations, such as geometries that we find more attractive. For example, we may search for solutions with such features using the solution enumeration capabilities of Z3 [18].

We plan to implement our solution technique in a hardware-based middleware controller that monitors back-end data systems, and reshapes incoming file traffic to build the proposed dynamic allocations of RAID groups in response to predictions for overprovisioning. We can also envision an extension enabling data storage system designers to query Z3 regarding hypothetical disk configurations and data dependence constraints as they design a new storage system, thus enabling them to optimize their designs with respect to the robustness/cost tradeoff before purchasing any hardware.

Availability

We have made our implementation, all associated source code, and data available under the terms of the University of Illinois/NCSA Open Source License[6] at our laboratory website `http://trust.dataengineering.org/research/nqueens/`.

Acknowledgements The authors would like to thank Nikolaj Bjorner, Rohit Dureja, and Varun Krishna for their helpful comments and corrections.

[6] `http://opensource.org/licenses/NCSA`

References

1 Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 399–410. ACM, 2013. doi:10.1145/2508859.2516718.

2 H. Peter Anvin. The mathematics of RAID-6, 2007.

3 Ulya Bayram, Eric William Davis Rozier, Pin Zhou, and Dwight Divine. Improving reliability with dynamic syndrome allocation in intelligent software defined data centers. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*, pages 219–230. IEEE Computer Society, 2015. doi:10.1109/DSN.2015.46.

4 Ulya Bayram, Kristin Yvonne Rozier, and Eric William Davis Rozier. Characterizing data dependence constraints for dynamic reliability using N-queens attack domains. In Javier Campos and Boudewijn R. Haverkort, editors, *Quantitative Evaluation of Systems, 12th International Conference, QEST 2015, Madrid, Spain, September 1-3, 2015, Proceedings*, volume 9259 of *Lecture Notes in Computer Science*, pages 211–227. Springer, 2015. doi:10.1007/978-3-319-22264-6_14.

5 Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, 2009. doi:10.1016/j.disc.2007.12.043.

6 Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185, 1994. doi:10.1145/176979.176981.

7 Peter F. Corbett, Robert English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar. Row-diagonal parity for double disk failure correction. In Chandu Thekkath, editor, *Proceedings of the FAST'04 Conference on File and Storage Technologies, March 31 – April 2, 2004, Grand Hyatt Hotel, San Francisco, California, USA*, pages 1–14. USENIX, 2004. URL: http://www.usenix.org/events/fast04/tech/corbett.html.

8 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

9 Alexandros G. Dimakis, Brighten Godfrey, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*, pages 2000–2008. IEEE, 2007. doi:10.1109/INFCOM.2007.232.

10 Daniel Duffy and John Schnase. Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. In *Proceedings of the 30th International Conference on Massive Storage Systems and Technology*. IEEE Computer Society, 2014.

11 B. Eickenscheidt. Das n-damen-problem auf dem zylinderbrett. *feenschach*, 50:382–385, 1980.

12 Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.*, 47:98–115, 2015. doi:10.1016/j.is.2014.07.006.

13 Casey Henderson. Usenix association fast 2013 memo, 2015. URL: https://www.usenix.org/system/files/conference/fast13/fast13_memo_021715.pdf.

14 Nathan Jacobson. *Lectures in Abstract Algebra: III. Theory of Fields and Galois Theory*, volume 32. Springer Science & Business Media, 2012. URL: http://www.springer.com/in/book/9780387901244.

15 M. C. Jones. Kumaraswamy?s distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, 6(1):70–81, 2009. doi:10.1016/j.stamet.2008.04.001.

16 David A. Klarner. Queen squares. *J. Recreational Math*, 12(3):177–178, 1979.

17 Vladimir Klebanov, Peter Müller, Natarajan Shankar, Gary T. Leavens, Valentin Wüstholz, Eyad Alkassar, Rob Arthan, Derek Bronish, Rod Chapman, Ernie Cohen, Mark A. Hillebrand, Bart Jacobs, K. Rustan M. Leino, Rosemary Monahan, Frank Piessens, Nadia Polikarpova, Tom Ridge, Jan Smans, Stephan Tobies, Thomas Tuerk, Mattias Ulbrich, and Benjamin Weiß. The 1st verified software competition: Experience report. In Michael J. Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods – 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2011. doi:10.1007/978-3-642-21437-0_14.

18 Ali Sinan Köksal, Viktor Kuncak, and Philippe Suter. Scala to the power of Z3: integrating SMT and programming. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23 – 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 – August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 400–406. Springer, 2011. doi:10.1007/978-3-642-22438-6_30.

19 Ponnambalam Kumaraswamy. A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, 46(1):79–88, 1980. doi:10.1016/0022-1694(80)90036-0.

20 Adam Leventhal. Triple-parity RAID and beyond. *ACM Queue*, 7(11):30, 2009. doi:10.1145/1661785.1670144.

21 Carl P. McCarty. Queen squares. *The American Mathematical Monthly*, 85(7):578–580, 1978. doi:10.2307/2320871.

22 Bernard A. Nadel. Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert*, 5(3):16–23, 1990. doi:10.1109/64.54670.

23 Jehan-François Pâris, Ahmed Amer, and Thomas J. E. Schwarz. Low-redundancy two-dimensional RAID arrays. In *International Conference on Computing, Networking and Communications, ICNC 2012, Maui, HI, USA, January 30 – February 2, 2012*, pages 507–511. IEEE Computer Society, 2012. doi:10.1109/ICCNC.2012.6167474.

24 Jehan-François Pâris, Darrell D. E. Long, and Witold Litwin. Three-dimensional redundancy codes for archival storage. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, USA, August 14-16, 2013*, pages 328–332. IEEE Computer Society, 2013. doi:10.1109/MASCOTS.2013.45.

25 David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In Haran Boral and Per-Åke Larson, editors, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988.*, pages 109–116. ACM Press, 1988. doi:10.1145/50202.50214.

26 Vera Pless. *Introduction to the Theory of Error-Correcting Codes, 3rd Edition*. John Wiley & Sons, 1998.

27 Eric W. D. Rozier and Kristin Yvonne Rozier. SMT-driven intelligent storage for big data. In *Proceedings of the Ninth International Workshop on Constraints in Formal Verification (CFV 2015)*, Austin, Texas, U.S.A., November 2015.

28 Eric W. D. Rozier and Kristin Yvonne Rozier. Cascading solution of data dependence constraints with Z3. In *Proceedings of the Fourteenth International Symposium on Artificial Intelligence and Mathematics (ISAIM 2016)*, Fort Lauderdale, Florida, U.S.A., January 2016.

29 Eric William David Rozier and William H. Sanders. A framework for efficient evaluation of the fault tolerance of deduplicated storage systems. In Robert S. Swarz, Philip Koopman, and Michel Cukier, editors, *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*, pages 1–12. IEEE Computer Society, 2012. doi:10.1109/DSN.2012.6263921.

30 Eric William David Rozier, William H. Sanders, Pin Zhou, NagaPramod Mandagere, Sandeep Uttamchandani, and Mark L. Yakushev. Modeling the fault tolerance consequences of deduplication. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, October 4-7, 2011*, pages 75–84. IEEE Computer Society, 2011. doi:10.1109/SRDS.2011.18.

31 Eric William David Rozier, Pin Zhou, and Dwight Divine. Building intelligence for software defined data centers: modeling usage patterns. In Ronen I. Kat, Mary Baker, and Sivan Toledo, editors, *6th Annual International Systems and Storage Conference, SYSTOR'13, Haifa, Israel – June 30 – July 02, 2013*, page 20. ACM, 2013. doi:10.1145/2485732.2485752.

32 Miguel A Salido and Federico Barber. How to classify hard and soft constraints in non-binary constraint satisfaction problems. In *Research and Development in Intelligent Systems XX: Proceedings of AI2003, the Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 213–226. Springer London, London, 2004. doi:10.1007/978-0-85729-412-8_16.

33 John L. Schnase, Daniel Q. Duffy, Glenn S. Tamkin, Denis Nadeau, John H. Thompson, Cristina M. Grieg, Mark A. McInerney, and William P. Webster. Merra analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. *Computers, Environment and Urban Systems*, 61, Part B:98–211, 2017. doi:10.1016/j.compenvurbsys.2013.12.003.

34 Thomas J. E. Schwarz, Darrell D. E. Long, and Jehan-François Pâris. Reliability of disk arrays with double parity. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2-4, 2013*, pages 108–117. IEEE Computer Society, 2013. doi:10.1109/PRDC.2013.20.

35 Rok Sosic and Jun Gu. Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Trans. Knowl. Data Eng.*, 6(5):661–668, 1994. doi:10.1109/69.317698.

36 Vernon Turner, John F Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *International Data Corporation, White Paper, IDC_1672*, 2014.

37 Eric W. Weisstein. Rooks problem, 2002.