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.

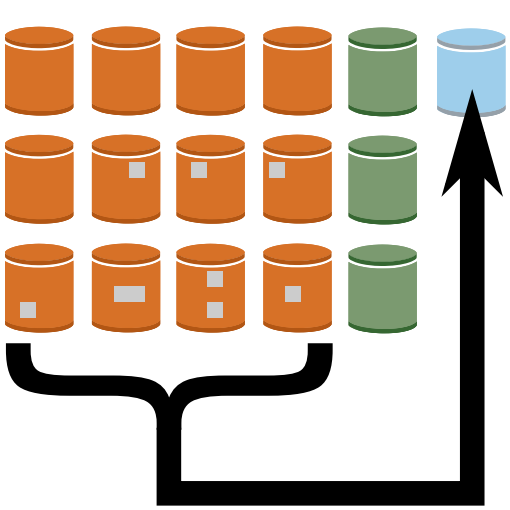
2.4 Novel Contributions

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.

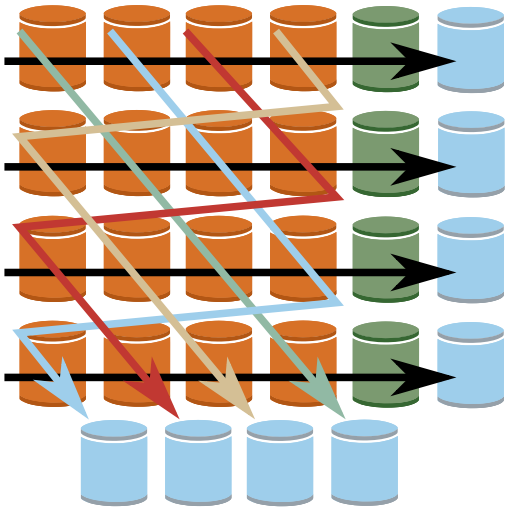
This paper is organized as follows: Section 3 provides background on dependence relationships in storage systems, and related work in novel RAID geometries. Section 4 introduces an encoding for this problem in a variant of n-queens, mapping the problem of data layout strategies that respect all data 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.

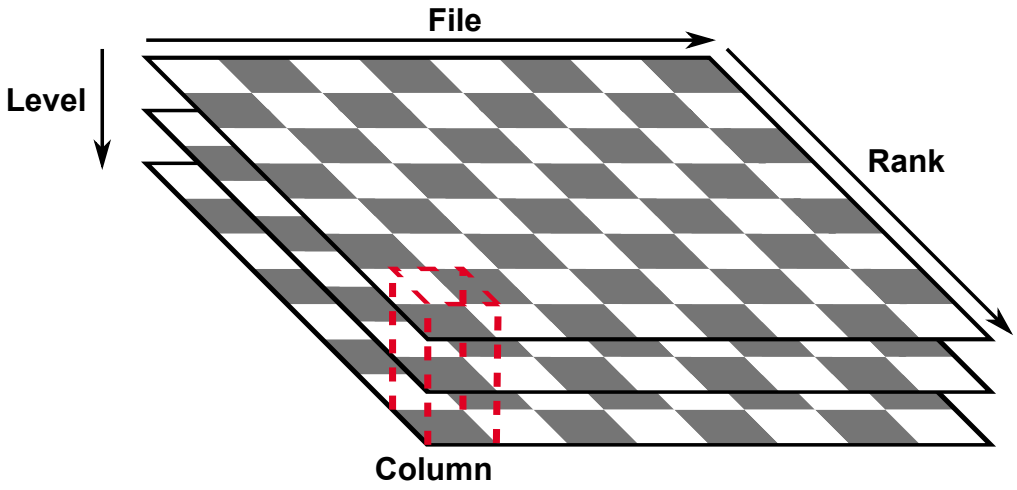


Figure 2 Example Space with dimensions 3x8x8.

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.

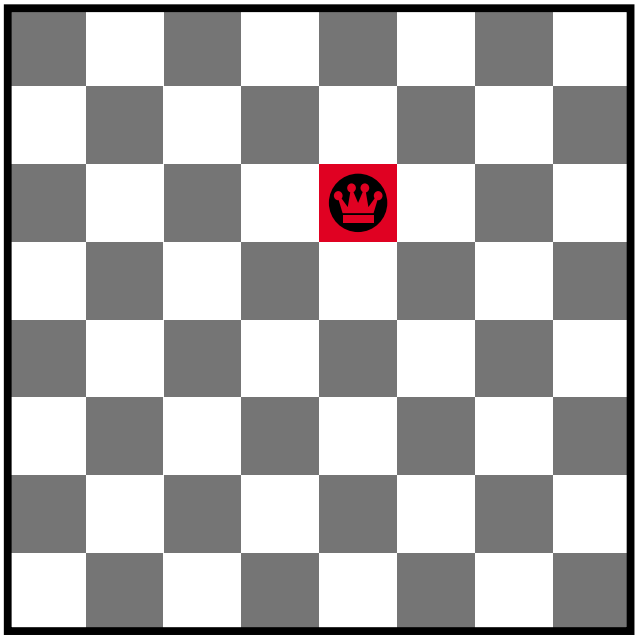


Figure 3 Example attack domain of a single degenerate queen.

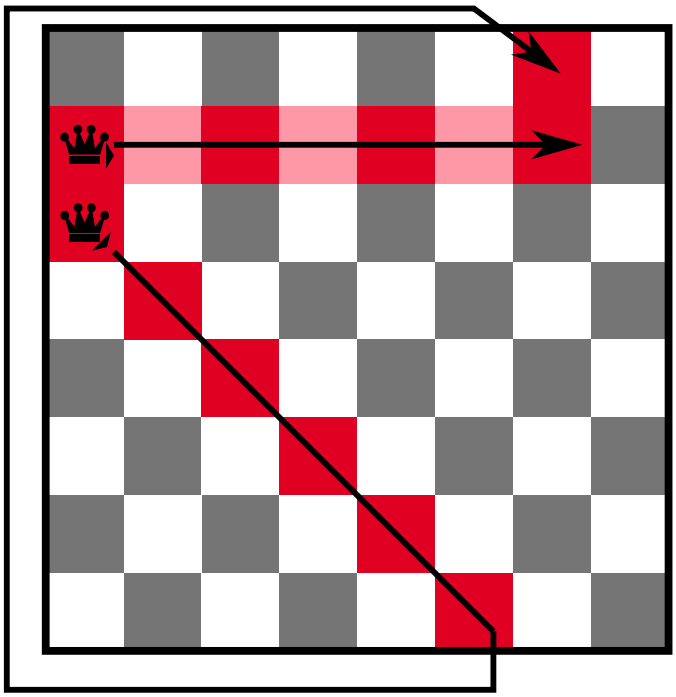


Figure 4 Example attack domains of two linear queens.

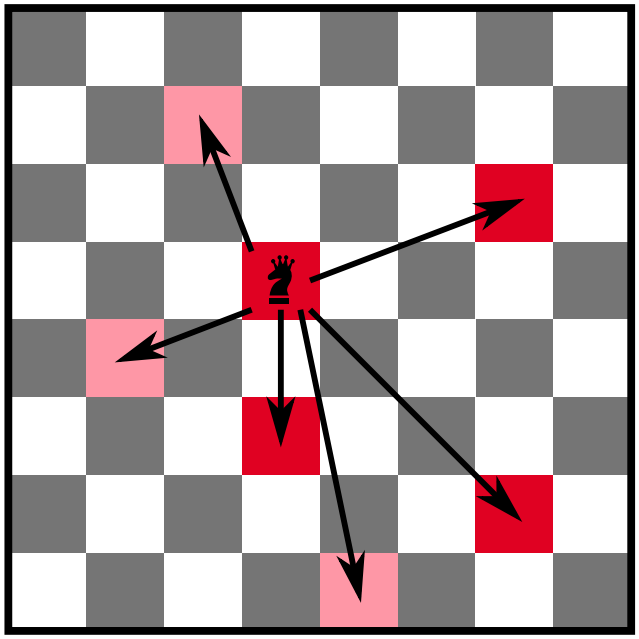


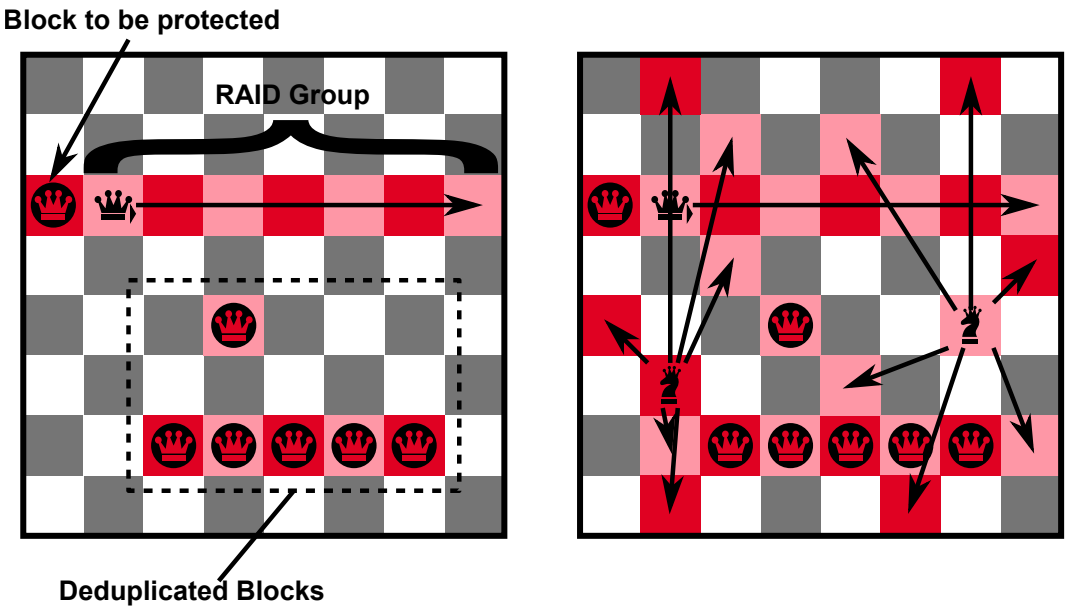
Figure 5 Example attack domain of a single indirect queen.

- **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.

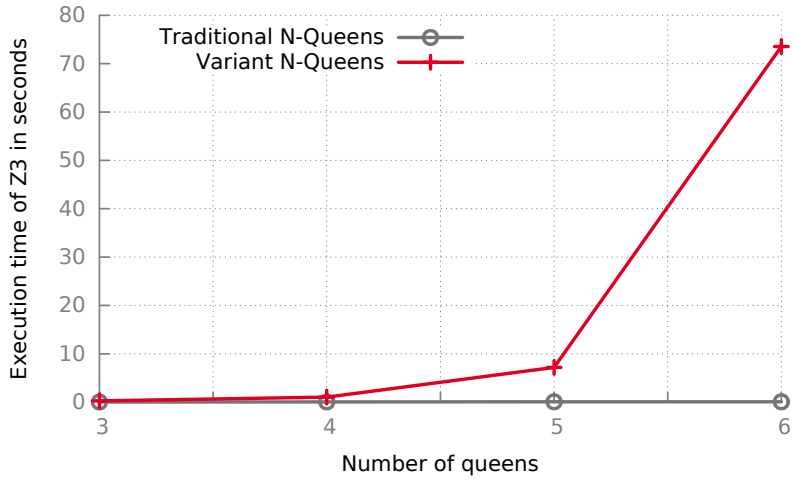


Figure 7 Comparison of solution times for Z3 given the placement of Y queens on a $Y \times Y$ traditional n-queens board and a $Y^2 \times Y \times Y$ variant n-queens board from our problem.

$$\forall l \in L, \forall r \in R, \forall f \in F(x_{l,r,f} \in \{\lambda, \Delta, \Lambda, \epsilon\}) \tag{5}$$

$$\forall l \in L, \forall r \in R, \forall f \in F((r = \lfloor l/F \rfloor) \rightarrow (x_{l,r,f} = \Delta)) \tag{6}$$

$$\forall l \in L, \forall r \in R, \forall f \in F((\exists b | b \mathcal{R} l) \rightarrow (x[l, r, f] = \Delta)) \tag{7}$$

$$\forall r \in R, \forall f \in F(p_{r,f} \geq \sum_{l} (x_{l,r,f} = \Gamma)) \tag{8}$$

$$\forall l \in L(\sum_{\forall r, \forall f} (x_{l,r,f} = \lambda) \geq W \cdot |F|) \tag{9}$$

$$\forall l \in L(\sum_{\forall r, \forall f} (x_{l,r,f} = \Lambda) \geq W) \tag{10}$$

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.

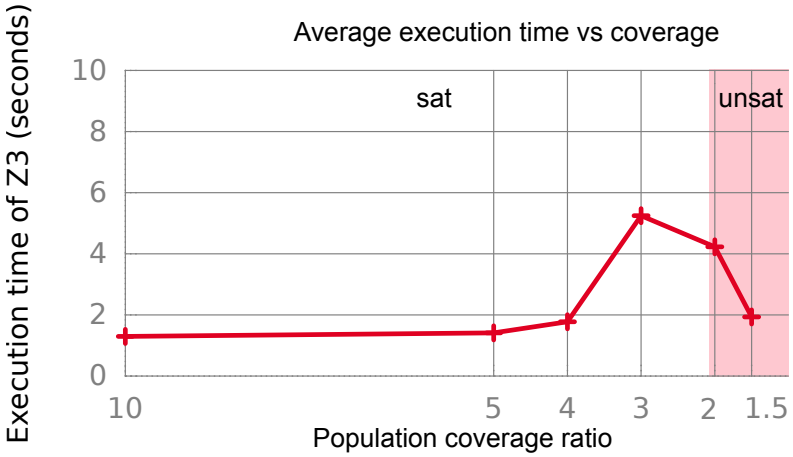
7 Experimental Results and Validation

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.

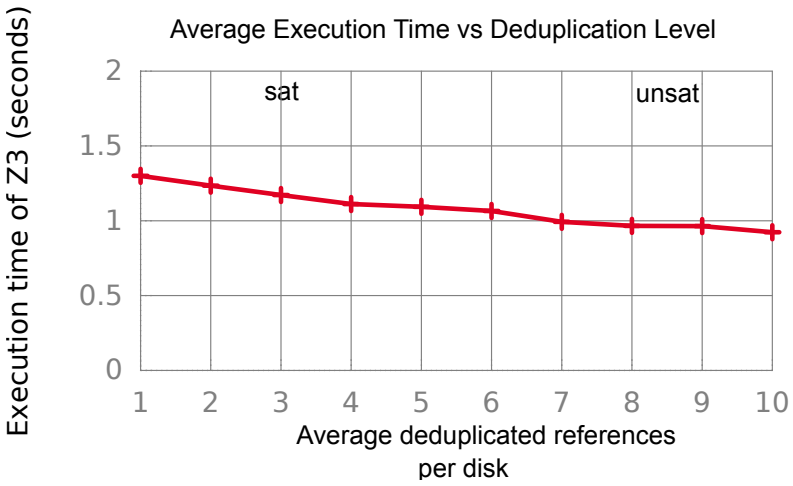
Figure 9 along with Figure 7 provide a summary of the results of our experiments. We found a sharp satisfiability cliff accompanying the population constraints board that correspond to the probability of a rank having no available space. This suggests an important observation to account for when moving 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

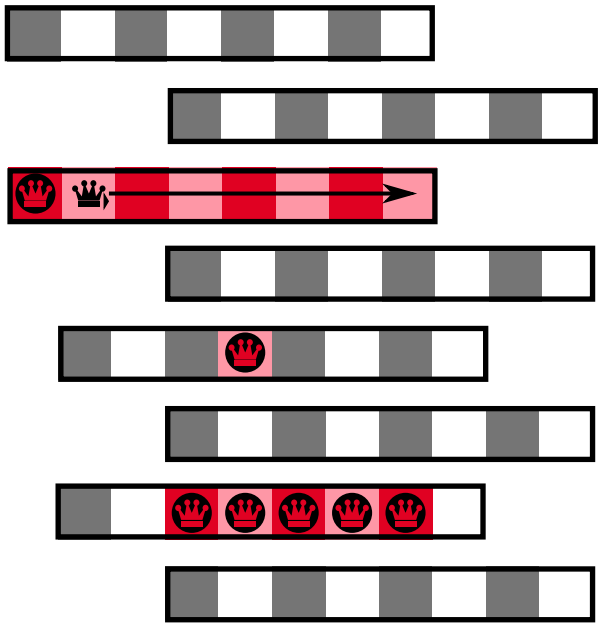


Figure 10 Example of row-wise decomposition.

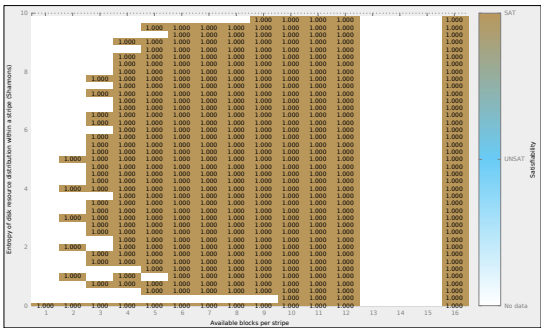
scale linearly with the size of our data center. We note the potential to further improve solution by partitioning the initial system on a row-wise basis, as shown in Figure 10. By sorting rows based on available resources for additional syndromes, and the distribution of those resources, we may be able 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.

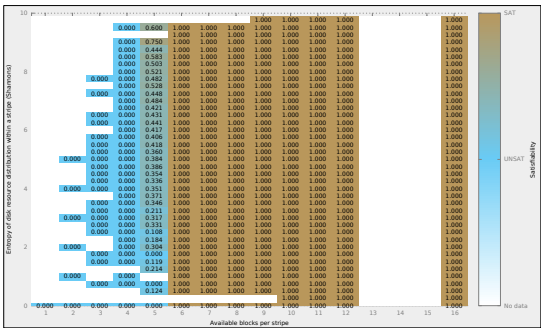
The run times of the resulting boards were likewise highly regular with respect to resources and their entropy, as shown in Figure 12. Again we note faster run times for higher numbers of available resources, and higher entropy distributions of those resources.

7.1 Application to Embedded and Resource Constrained Systems

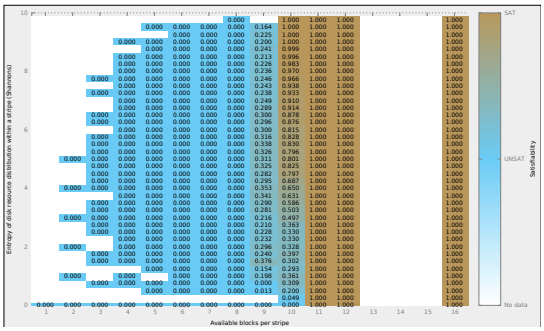
The results of the experiments shown in Figures 11 and 12 describe a highly regular space can be generated when our problem is characterized in terms of available resources and entropy. This space is learnable for problems of a given size via estimation through a random sampling of the feature space 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.



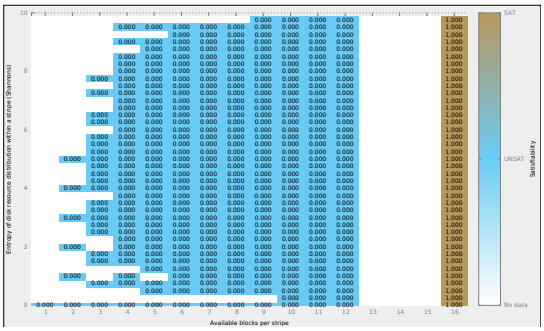
(a) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 1$.



(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 $W = 3$.

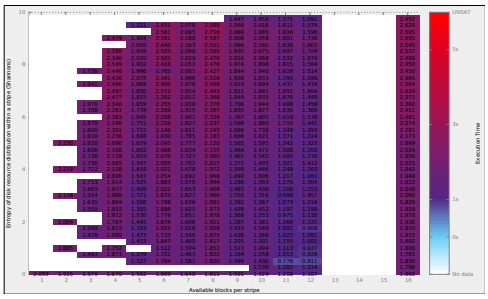


(d) Satisfiability of rows based on available resources, and the entropy of resource distribution for $W = 4$.

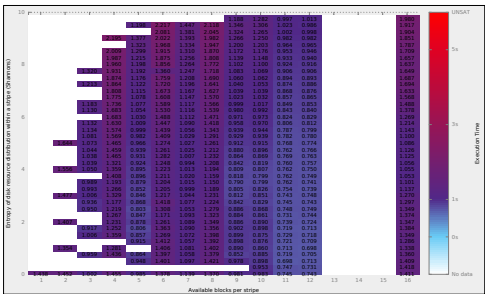
Figure 11 Probability of a satisfiable solution for $W \in [1, 4]$ based on available resources, and the entropy of resource distribution.

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.

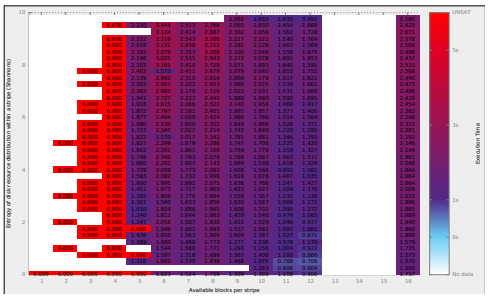
By generating this table in advance we can apply our technique selectively by first choosing some protection level that our tables indicate is solvable for a system with the currently observed resource levels and resource entropy, and solving that problem first. If time remains before our deadline is 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.



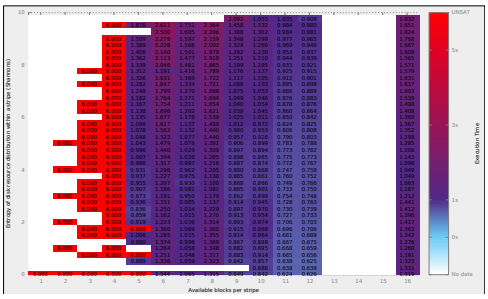
(a) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 1$.



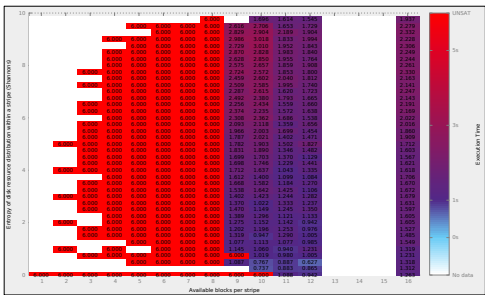
(b) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 1$.



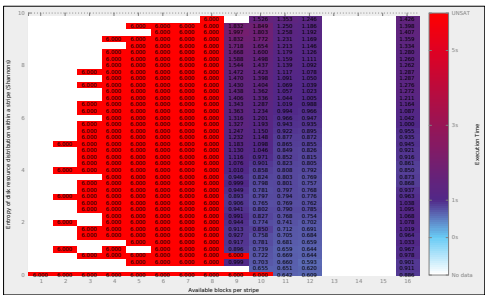
(c) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 2$.



(d) Mean solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 2$

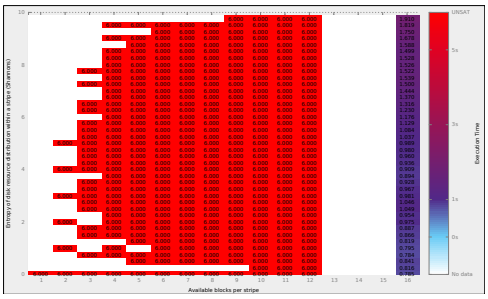


(e) Maximum observed solution time of Z3 based on available resources, and the entropy of resource distribution for $W = 3$.



(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.