Evaluation of Meta-scheduler Architectures and Task Assignment Policies for High Throughput Computing

Eddy Caron, LIP/ENS Lyon, 46 Alle d'Italie, 69364 Lyon Cedex 07
V. Garonne, A. Tsaregorodtsev
Centre de Physique des Particules de Marseille, Marseille, France

January 29, 2006

Public Note

Issue : 1
Revision : 0
Reference : LHCb-2005-082-Offline
Created : January 12, 2005
Last modified : January 29, 2006

Abstract

In this paper we present a model and simulator for many clusters of heterogeneous PCs belonging to a local network. These clusters are assumed to be connected to each other through a global network and each cluster is managed via a local scheduler which is shared by many users. We validate our simulator by comparing the experimental and analytical results of a M/M/4 queuing system. These studies indicate that the simulator is consistent. Next, we do the comparison with a real batch system and we obtain an average error of 10.5% for the response time and 12% for the makespan. We conclude that the simulator is realistic and well describes the behaviour of a large-scale system. Thus we can study the scheduling of our system called *DIRAC* in a high throughput context. We justify our decentralized, adaptive and opportunistic approach in comparison to a centralized approach in such a context.

0.1 Introduction

In an institutional large scale system [5] resources are heterogeneous clusters which belong to a local network (LAN) and miscellaneous administrative domains. These clusters are shared between many users or virtual organizations [6] and a local policy is applied to each cluster which defines their access rights. This policy is applied through a resource management system, *i.e.* a batch system.
To aggregate theses clusters and manage the workload a global architecture must be defined paying special attention to the size of these systems. For example, in the High Energy Physics domain the envisaged size is around a hundred sites spread all over the world which is equivalent to 30,000 nodes. This domain also strongly favors *high throughput* [12], which attempts to maximize the number of jobs completed on a daily, or longer, basis. This is typical of situations where the supply of computational jobs greatly exceeds the available resources and jobs are generally not time critical. While batch systems are often used at the local level, there is no common solution in a global context. The *DIRAC* system has been developed to meet these requirements and provide a generic, robust grid computing environment.
In this paper we propose an evaluation of the performances and the behavior of the *DIRAC* system in a *high throughput context*. The paper is organized as follows: Section 2 presents the background; Section 3 discusses the DIRAC architecture and main components; Section 4 presents the model used; while Section 5 discusses the simulation tool and section 6 descibes how it was tested; Section 7 shows the experimental setup; Section 8 the results and finally Section 9 finishes with conclusions and future plans.

0.2 Background

In a multi-site grid project [18, 8, 1], decisions are often taken with a global view of the system. The architecture in Figure 1 is composed of a centralized meta-scheduler and a centralized information system. This approach put in place as 'Push' paradigm.

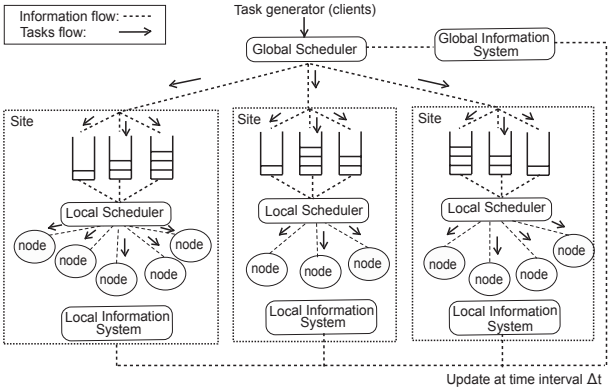


Figure 1: Example of an architecture with centralized scheduling

In this model, the global information system keeps all the static and dynamic information about the system state in one place. Sensors deployed on the sites update the information by first querying the local information system and then updating their own information in the global information system. Ideally these updates are done whenever a change of state happens in the system which could, for example, be the arrival or end of a task. In fact, this solution often generates a message storm and needs some kind notification mechanism. The use of a period $\triangle t$ seems most appropriate and stems the flow of messages.

Some studies [9] propose strategies which employ file queuing systems [10] whilst others [17] use simulation mechanisms like BRICKS [16]. Generally these work quite well but only in a simplified and unrealistic model. Thus far no project is able to manage the workload on more than a hundred sites and the problem of a multi-site system for high throughput computing has not yet been explored.

0.3 The *DIRAC* system

DIRAC (Distributed Infrastructure with Remote Agent Control) [7] has been developed by the CERN LHCb physics experiment to facilitate large scale simulation and user analysis. The *DIRAC* system has recently been used for an intensive physics simulation involving more than sixty sites, 90 TB of data, and in excess of one thousand 1 GHz processor-years. *DIRAC* is organized into a *Service Oriented Architecture* (SOA), with a number of independent services including monitoring and resource management.

0.3.1 The resource management system: The 'Pull' paradigm

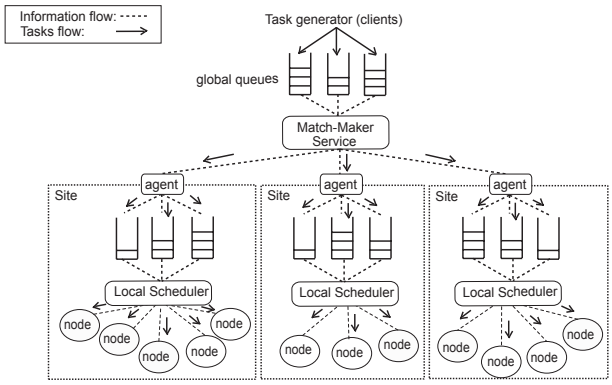


Figure 2: The *DIRAC* scheduling model.

Figure 2 illustrates the *DIRAC* scheduling model which deploys agents on the sites and uses central global queues. *DIRAC* uses a *'Pull'* paradigm where agents demand a task if they detect free slots. Using the cycle-stealing paradigm borrowed from global computing [14], tasks are only run when resources are not in use by the local users. *DIRAC* extends this concept to different computing resources by defining a criterion of availability. These resources could be anything from a simple PC to whole batch systems. As soon as a resource is detected to be available the dedicated agent requests tasks from the `match-maker` service. This is accomplished using the resource description which contains the dynamic and static information about the resource. The `match-maker` service allocates tasks to resources by viewing the global job queues and using *Classad* matchmaking from the *CONDOR* project [12].

The Matchmaker compares one-on-one requirements performing a round-robin on each of the job queues until it finds a suitable job for a particular resource. This is an *O(n)* operation, which, in the worst case, would involve all *n* queued tasks in the system being checked once against the resource characteristics defined in the task request. This operation is independent from the total number of resources and the total number of tasks.

0.4 The performance model

Let $\mathcal{C}$ represent the set of clusters present in the multi-sites platform. Each cluster $\mathcal{C}_i$ owns a set of worker nodes $\mathcal{N}_i$ and belongs to a local domain, i.e. a LAN (Local Area Network). This local network describes a graph for the nodes. Each link of this graph has **a local bandwidth** $bwt_{\mathcal{C}_i}$ and **a local latency** $latency_{\mathcal{C}_i}$.

0.4.1 The topology

A cluster $\mathcal{C}_i$ is connected to the global network or WAN (World Area Network) by a switch. Figure 3 describes this topology with links having the same properties as previously mentioned. Let $bwt_{\mathcal{C}}$ be the **global bandwidth** and $latency_{\mathcal{C}}$ **the global latency**.

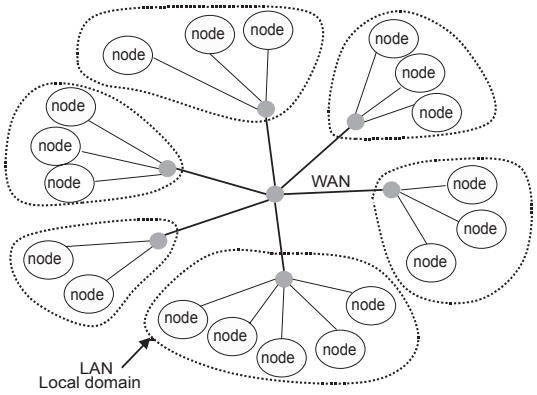


Figure 3: Example of a topology for a meta-scheduling platform.

Many different approaches exist to generate the right graph for the proposed model. Tools exist, such as ENV [15] to describe the characteristics of a real topology. Although there is not sufficient information to suggest that these tools scale well it was decided to use a generator topology. Some recent studies [13] show that networks follow specific power laws. The graph generator according to these laws are generally random, degree-based or hierarchical.

0.4.2 The node characteristic

Let (i, j) be the pair defines the j^{th} node of cluster $\mathcal{C}_i$. Each node (i, j) has a processor capacity $capacity_{i,j}$ and to express this we define one computing unit, the ***NCU*** (Normalized Computing Unit). This unit is determined by special application benchmarking on different referential machines, taking into account the absolute time. So the capacity of a node is simply the total number of computing units able to be computed per unit time. We can then model the platform heterogeneity and define **the average platform capacity** as $capacity_m = \frac{1}{\sum_{i \in \mathcal{C}} card(\mathcal{N}_i)} \sum_{i \in \mathcal{C}, j \in \mathcal{C}_i} capacity_{i,j}$.

0.4.3 The workload model

We define two levels for the workload, local and global. The global workload corresponds to the tasks submitted to the metacomputing system, usually called meta-tasks. The local or background workload corresponds to tasks locally submitted to a cluster. A meta-task m_k is mapped locally to a simple task k.

A typical task k has four attributes : $attributes$=$\{tl_k, length_k, proc_k, group_k\}$ where tl_k is the **local submission date**, $length_k$ **the length** expressed in NCU, $proc_k$ is **the total number of processors** required for the task execution and $group_k$ **the organization** who submits the task. A meta-task mk is composed of the task properties sub-set and **the global submission date** t_k. So we have $meta\text{-}attributes_k$=$\{t_k, attributes_k\}$.

Modelling the workload for a metacomputing system involves determining k for each task from the task set $\mathcal{T}$ then submitting the attributes $attributs_k$ to a cluster C_i. The meta-tasks mk of the set of meta-tasks $\mathcal{MT}$ and their meta-attributes are also submitted to the system. The methods used to generate a workload are the following: a randomize workload; a workload derived from real system traces and lastly a stochastic workload.

A random workload and, in the same way, a workload derived from a real system are not realistic. A workload derived is judged to represent too many platform dependant characteristic and so too specific. Instead, the stochastic workload is chosen here. Some works [11] studying computing centre traces propose a complete probabilistic model and so we write $S(\mathcal{T})$ for **the distribution function which generates the length set** for a set of task $\mathcal{T}$. Let CA be **the cut applied** to this length set which fixes the maximal and minimal length. We also denote **the distribution function which generates the submission date set** by $A(\mathcal{T})$ for a set task and finally **the average submission rate** by $\lambda_{\mathcal{T}}$.

0.4.4 The local model scheduling

At the local level, nodes of a same site are typically managed by a resource management system, e.g a batch system. Other implementations use queues which are defined by the characteristics of the task, for example, their length. Shared time scheduling between users is done by the local scheduler which would normally apply policies based on quotas or priorities. For a cluster $\mathcal{C}_i$, we can define **a queues set** $\mathcal{Q}_i$. Each queue $\mathbf{q_{i,j}}$ of $\mathcal{Q}_i$ is composed of a set of nodes $\mathbf{N_{q_{i,j}}}$. Any particular node can belong to one or many queues. The tasks submitted to the site are then added to these queues to wait for their execution. Subsequently, a queue $\mathbf{q_{i,j}}$ will contain a task set $\mathcal{T}_{i,l}$. We define **the queue depth** $depth_{i,j} = card(\mathcal{T}_{i,l})$ as **the total number of tasks waiting** in the queue a particular instant. The *maximal time* that a task could spend in execution on a node of the queue $q_{i,j}$ is denoted by $t_{max_{i,j}}$.

0.4.5 Measures and metrics

For a task k, we define three following states: *queued*, *running* and *done*. The state *queued* means that a task is in a waiting queue. When the task is executing it is in the *running* state and the *done* state signals that the task is completed. **The corresponding times for the changing states** *running, queued* **and** *done* **for a task** k **are respectively** r_k, q_k **and** d_k. **the local waiting time for a task** k is the execution beginning time minus the submission time $r_k - tl_k$. **The execution time** is $d_k - r_k$ and **the local response time** is $d_k - tl_k$. For a meta-task mk, we have a **global waiting time** which is the beginning execution time minus the global submission time, $r_k - t_k$. **The global response time** is $d_k - t_k$.

For the set of meta-tasks $\mathcal{MT}$, we define **the average waiting time**:

$$waiting_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (r_k - t_k) \tag{1}$$

the average execution time:

$$execution_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - r_k) \tag{2}$$

and **the average response time**:

$$response_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - t_k) \tag{3}$$

We also define **the makespan** which is the full time to complete all the jobs in $\mathcal{MT}$:

$$makespan = \max_{k \in \mathcal{MT}}(d_k) - \min_{k \in \mathcal{MT}}(t_k) \tag{4}$$

0.5 Simulation tool

Simgrid [2] is a discrete event toolbox which allows the modelling and description of a platform for centralized, hierarchical or fully distributed scheduling. The improvements made to *Simgrid* are as follows:

A description platform module. Our simulator is interfaced with the hierarchical graph generator *Tiers* [4]. We have to specify the total number of WAN, LAN, nodes per LAN and the redundancy links. For the capacity information, we define a sample set of nodes where each node is weighted by a percentage. This percentage expresses the proportion of this node type present in the platform. The node NCU capacities and their weights are inspired by the performances obtained by *DIRAC* for a physics application on the production platform [3]. This platform was composed of more than 4,000 nodes and twenty different node configurations. Based on the total number of nodes and their proportions we generate the set $\mathcal{P}$ of all available capacity. Then for each node we proceed by drawing lots in the set $\mathcal{P}$ and one occurrence of this value would be removed from the set $\mathcal{P}$ until the capacity attribute is filled for all nodes.

A workload generator. *Simgrid* has already got an implementation of the task concept. In this model, however, the meta-data is added *e.g.,* the organization submitting tasks. The workload generator provides different probability density functions (p.d.f) like the Gamma law, Gaussian law and so on. To have a shared system we implement an agent per client or organization. The simulation tool allows one to simulate different system users. Therefore, it is possible to have different workloads submitted at the same time and evaluate their interactions.

A generic batch system. The basic entity at the local level is the batch system. As *Simgrid* does not provide a model for this, a generic one has been implemented. The design is illustrated by Figure 4.

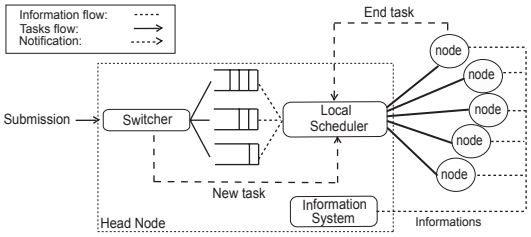


Figure 4: The generic Batch system design.

A head node hosts the principal components: switcher, queues, information system and finally the local scheduler. Each node communicates with the head node. A task submission is managed by the switcher which with regard the task requirements, places it in a queue and notifies the scheduler. This then queries the monitoring and accounting system to choose a candidate node. If no resource is available the task stays in the queue but once the task is sent to the node it is executed. After the task is completed, the scheduler is notified which triggers a cycle where the scheduler looks at the queues and determines if another task could be executed. The scheduler configuration is entered by file and includes properties such as the total number of queues, the availability or not of a node in a queue and the maximum number of tasks which could execute on a node.

The meta-scheduling architectures. Two kinds of global architectures were implemented. First the centralized architecture outlined in Section 0.2. Second the *DIRAC* architecture described in the Section 0.3.

A monitoring and accounting system. For each simulation run the information relating to each task is recorded. This helps the analysis of a particular strategy by facilitating the measures and metrics computation described in Section 0.4.5.

0.6 Validation of the simulation tool

0.6.1 Analytical validation

For the theoretical validation, experiments were performed on M/M/m queuing systems [10].

Figure 5 shows the response time differences between the simulated results and the analytical theory for a M/M/4 system. The service time average is four units time. The arrival rate follows an exponential law. The simulated responses are derived from 16 independent runs of 1,000 tasks and the root mean square error for all simulated arrival rates. The results obtained are consistent with theory.

0.6.2 Experimental Comparison

A dedicated and heterogeneous cluster was used, described in Table 1.

A *DIRAC* agent was deployed on the cluster with a task generator put in place. This generator submits independent and sequential tasks with no communications. The submission times follow a Poisson law and

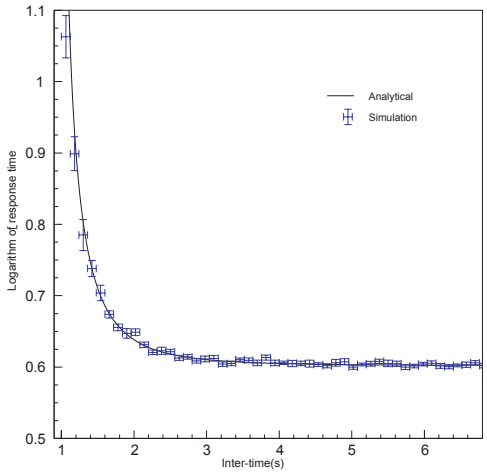


Figure 5: Reponse time comparison between simulation and M/M/4 theory queueing system.

Attributes	**Values**		
Total number of nodes ($card(\mathcal{N}_i)$)	3		
Type	PII	PII	PIII
Processor(MHz)	350	400	600
Memory(mo)	128	128	128
Capacity($NCU.s^{-1}$)	32.12	52.12	100.00
Scheduler	openPBSv2.3		
Politicy	First Come First Serve(FCFS)		
Local Network	Megabyte Ethernet		

Table 1: Platform caracteristics used for the simulation validation.

the benchmark used was a program which implements a CPU consumed counter. It takes one parameter which is the number of CPU to consume before ending. This length is created for each task and follows a Weibull law. The response time and the waiting time are then captured by the *DIRAC* monitoring service. Then, we capture this workload to inject it in our simulator. To estimate the execution time according to the node capacity we normalize this time with the node NCU capacity. The NCU node capacity is determined by benchmarking, outlined in Table 1. A simple topology is assumed where each node is connected to the head node by a simple link with 100 mega byte/s bandwidth and a null latency, as illustrated in Figure 4 .

0.6.3 Results

The total number of tasks is 330, i.e. $card(\mathcal{MT}) = 330$. We observe for first instance an average error of 80% for the response time as illustrated in Figure 0.6.3. After a trace study we characterize two service times, μ_{rec} and μ_{send}. μ_{rec} is the service time between task arrival and task sending on a node or in queue and μ_{send} is the necessary time to notify the scheduler of a task completion. This large average error can be explained by the fact that the scheduler made its choice with a different system state view. The nodes are heterogeneous so the consequences are dramatic for the response time. We correct this error by including the service times μ_{rec} and μ_{send} measured on the real system injected as traces. Then, we obtain exactly what we would expect in reality which validates the code. The experiment is then repeated by setting time services to constants. These constants are the average service time observed in reality ($\mu_{rec} = 3.75s$, $\mu_{send} = 2s$). After this we observe an improvement in the average error of 10.5%. Figure 0.6.3 shows the makespan evolution versus the total number of tasks. For the constant service time, we see an average error of 12%. From this it is possible to conclude that at the local level the simulator is realistic. It is now possible to proceed to the strategies and meta-scheduling architectures evaluation. One further improvement could be to make the service times μ_{rec} and μ_{send} a distribution function approaching the real behavior.

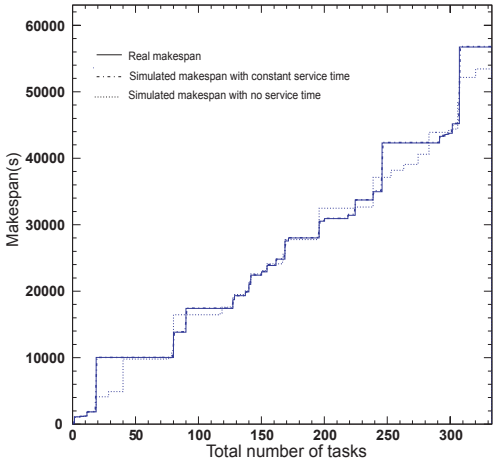


Figure 6: Comparison between simulation and a real batch system with $\mu_{rec} = \mu_{env} = 0$ and $\mu_{rec} = constant$, $\mu_{env} = constant$. Makespan evolution vs. the number of tasks.

0.7 Experimentals setup

The decentralized *DIRAC* architecture described in Section 0.3 and the centralized approach from Section 0.2 can now be compared. The message control size for both architectures is 30 KB in the simulation. The workload characteristics are inspired by an empirical study [11] and Table 2 summarises the platform parameters and workload attributes.

		Parameters	*Notations*	*Values*
Platform		Total number of sites	$card(C)$	3
		Total number of node per site	$card(N_i)$	20
		Total number of queue per site	$card(F_i)$	1
		mean node capacity	$capacity_m$	96 $NCU.s^{-1}$
		Local policy	M/M/$card(N_i)$/FCFS	FCFS
		Maximal execution time	$t_{max_{i,j}}$	24000s
		local/global bandwith	bwt_C/bwt_{C_i}	1000 $Mbit$/100 $Mbit$
		local/global latency	$latency_C$, $latency_{C_i}$	0s
Workload		Task Type	$card(proc_k)$	1
		Length distribution	$S(\mathcal{M}t) \rightarrow \{size_k\}$	$Weibull(\alpha = 142.2, \beta = 0.45)$
		Length cut	$\mathcal{C}(taille_k)$	$37300 < taille_k < 242800$
	Global	Total number of task	$card(\mathcal{M}t)$	500
		Arrival time distribution	$A(\mathcal{M}t) \rightarrow \{tl_k\}$	$Poisson(m = 0.05, s = 4)$
		mean inter-arrival	$1/\lambda_{\mathcal{M}t}$	19s
	Local	Total number of task per site	$card(\mathcal{T}_i)$	500
		Arrival time distribution	$A(\mathcal{T}_i) \rightarrow \{t_k\}$	$Poisson(m = 0.011, s = 4)$
		mean inter-arrival	$1/\lambda_{\mathcal{T}_i}$	87s

Table 2: Experiments parameters.

The associated *DIRAC* strategy is detailed in section 0.3. The criterion of availability is expressed in 5 which implies that tasks in the waiting state scheduled on a computing resource should not exceed 30% of the total number of nodes.

$$\frac{depth_{i,j}}{card(\mathcal{N}_{q_{i,j}})} < \varepsilon,\ e.g.\ \varepsilon = 0.3 \tag{5}$$

The policy applied at the matchmaker level is that of FRFS(Fit Resource First Serve). That means that the first resource which matches well is chosen.

We also propose to evaluate the impact of the deployment in *DIRAC*. Let us consider two kinds of agent

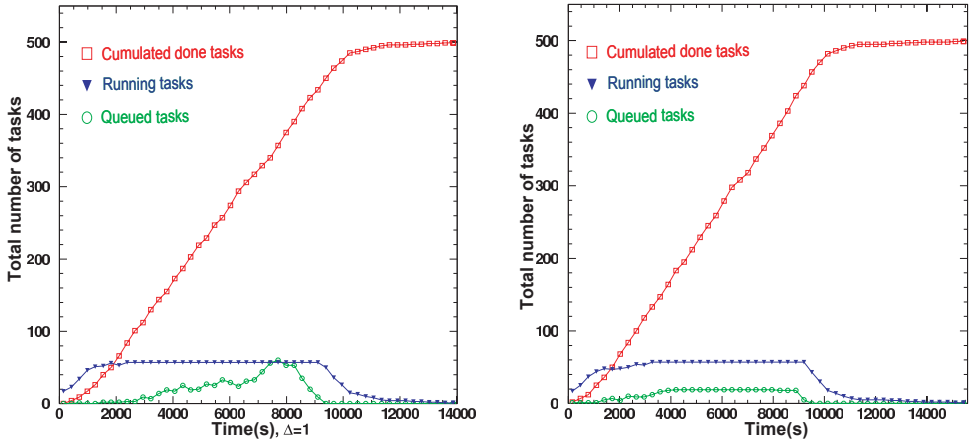
deployment. The static approach is described in Section 0.3 whereas the dynamic approach is a concept similar to the resource reservation. After detecting the availability, the agent deployed on the site queries the match-maker to ask if tasks are available. In the case of a positive answer, it submits an agent wrapped in simple task to the cluster. Once the agent arrives at the node, it checks the node capacity and environment and creates the resource description accordingly. After that the agent queries a task from the match-maker. If no task is returned the agent dies. In the simple reservation mode 'Run Once', the agent dies after the completion of the first task while in a 'filling' mode it queries for one more task with respect to the available time.

The algorithm for the centralized scheduling is the following. At each task arrival the scheduler looks for the least loaded resource, *i.e.* the resource $q_{i,j}$ from cluster C_i which has the minimum measured depth $depth_{i,j}$ with $\forall i \in \mathcal{C}$ and $\forall j \in \mathcal{F}_i$.

Now we compare two approaches which strongly favour high throughput computing but the question is: what architectures and implementations could influence their performance?

0.8 Results

Figure 7 shows the evolution of total number of tasks in the state *queued* and *running* during the experiment. The third line is the *done* cumulated task curve.



(a) Centralyzed approach with $\triangle t = 0$.

(b) *DIRAC* approach.

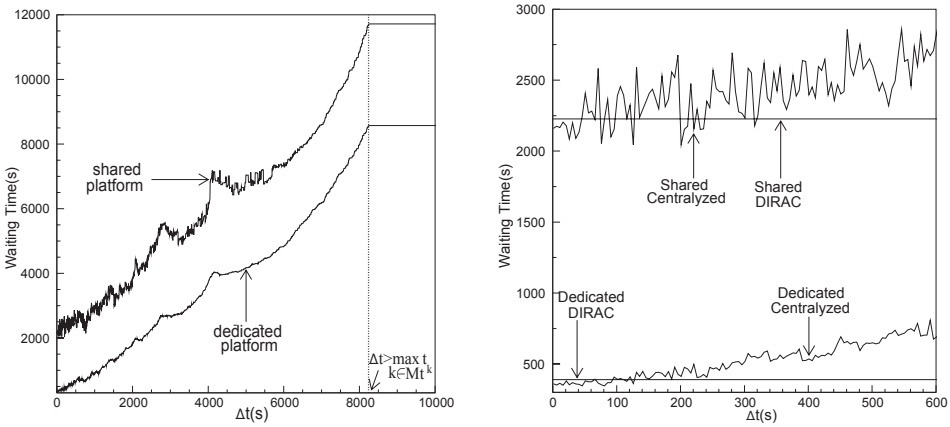
Figure 7: Tasks evolution vs. time in a dedicated platform.

The saturation phase gives us the platform maximal capacity which is equivalent to the sum of all nodes ,*i.e.* $\sum_{i \in \mathcal{C}} \mathcal{N}_i$, here 60. The two approaches saturate all resources but this is different in the *DIRAC* approach where the evolution of the tasks in the *queued* state is constant (Figure 7(b)).

The Figure 8 shows the variation of the $\triangle t$ period versus average waiting time $waiting_m$ for centralized scheduling in first a dedicated context and then in a shared context. The *DIRAC* waiting times are qualitatively indicated because they are independent in philosophy from $\triangle t$ (Figure 8(b)).

DIRAC does not use a central information service so does not depend on this period. For a $\triangle t$ less than 95 s, the waiting time is better than the centralized scheduling in a dedicated context and performs better by around 60 s in the shared context. In the latter the performances rapidly degrade and a more chaotic effect is observed. The upper bound observed corresponds to the situation where all tasks are scheduled on the same site where $\triangle t > \max_{k \in \mathcal{M}t} t_k$.

Figure 9(a) compares the makespan as well as the local and global response times executed for the four evaluated strategies. For a null $\triangle t$, the best makespan is obtained for the centralized approach although the smallest response time came from the *DIRAC* approach in the *filling* reservation mode. The execution times are of the same order for all strategies. This is explained by the platform characteristic that sites have



(a) Full variation.

(b) Comparison with *DIRAC*.

Figure 8: Average waiting time for meta-task vs. period $\triangle$t for the centralized approach in a shared and a dedicated context.

the same capacity on average. The response time difference is mainly due to the local and global waiting time. The largest local waiting time is found with the static *DIRAC* scheduling however, the global waiting time in this situation is minimal.

In the case of *DIRAC* reservation the local waiting times are null because the matching is done directly from the node. The waiting time is expressed for the agents in this case. The effect of changing the deployment from static to dynamic gives a improvement of 10% for the average response time. The reservation mode *filling* nearly introduced a 50% improvement for the average response time in comparison with the centralized approach.

The graphs in Figure 9(b) illustrate the rate of tasks demanding `the match-maker service` in the tree *DIRAC* deployment mode with the static approach (top), the simple reservation (middle) and the 'filling' reservation mode (bottom).

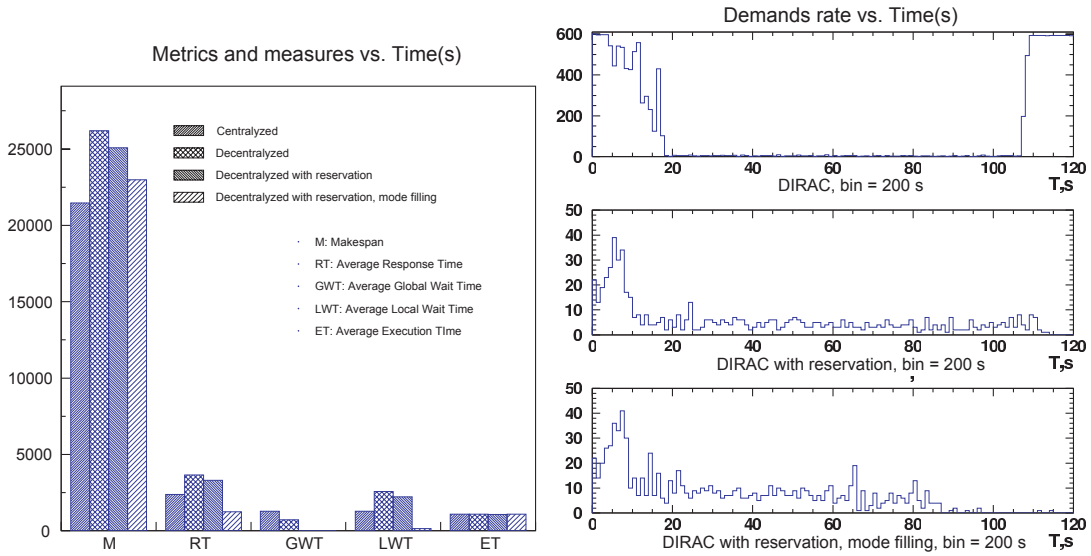
During the platform initialization and at the experimentation end, the load is important on `the service matchmaker` in the static case. The total number of queries with the static approach is about 872 with about 699 for the dynamic case. We also establish that the `the match-maker` load is more homogeneous for the dynamic case.

0.9 Discussion

In an ideal situation the centralized approach gives the best results but it is often impractical to assume that such a platform would stay stable. Common failures, by order of importance are: network failure; the disk quotas; unavailability of services; incorrect local configuration and finally power cut. With this large scope of error it is difficult to keep an ideal view of a global system. The scheduling is totally dependant on the information system performance and this system often does not scale well.

DIRAC bypasses this problem because one of the main characteristics is the total absence of a system global view. It takes its decisions with a partial and global view. Each resource, in conjunction with its current state, gets an appropriate workload to suit its capacity. The tasks are put in a buffer where the scheduling event is an attribute of the resource availability which is the opposite to the centralized approach where the triggered event is a task submission.

If a platform deterioration occurs, any drawback from using the centralized approach is immediately paid back in term of performance. This effect is also more significant if the approach is combined with predictions. A rapid state change of a resource is taken into account only after a lapse of time in the centralized model. During this lapse in a high throughput context, the decisions made can be disastrous. Resource starvations and system information failures are also two main drawbacks which do not affect *DIRAC*, where



(a) Strategies comparison for: centralized approach, *DIRAC* with a static deployment and the two *DIRAC* reservation mode simple and filling.

(b) Traffic on the match-maker service vs. time for the three *DIRAC* strategies.

Figure 9: Results and caracteristics for strategies.

all available resources are utilized immediately.

DIRAC demonstrates adaptability. This dynamic aspect forces scheduling in an opportunistic, reactive and non-predictive way. On the other hand, the results are quite similar with the centralized scheduling. *DIRAC* is easy to implement, stable and flexible. It also facilitates resource reservation which can significantly increase performances. Nevertheless it must be said that technically this improvement required direct communication with worker nodes. Within *DIRAC* passive communication mode relaying of outbound connectivity is used to accomplish this.

The reservation mode causes a higher and more regulated load on `the match-maker service`. This penalty for this improvement is the huge number of agents which abort right after the non-task answer (299 in our case which is non-negligible).

0.10 Conclusion and future works

In this paper we propose a model for a meta-scheduling platform. We measure an average error of 12% for the makespan prediction. With this tool it is demonstrated that a centralized approach is better than a decentralized approach in term of performances for high throughput computing. However, this happens only in the ideal case where the update period is quasi null. Above 95s in a dedicated context, the 'pull' approach had similar results and importantly was more stable. The same observation is made in a shared context. The 'pull' approach also provides an abundance of scenarios which allow a performance enhancement the of just under fifty percent compared to the centralized approach. This was most evident with resource reservation. It will be interesting to study the impact of the migration from site to site with regard to their local workload. The criterion for optimizing the scheduling is specific to the application itself, therefore, since many applications are executed concurrently on the same platform, grid scheduling must be done with a multi-criteria scheduling approach. Future work will be into the study of this aspect.

Acknowledgment

We gratefully acknowledge Stuart Paterson for helping to prepare this document.

Bibliography

[1] Abhijit Bose, Brian Wickman, and Cameron Wood. Mars: A metascheduler for distributed resources in campus grids. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 110–118, Washington, DC, USA, 2004. IEEE Computer Society.

[2] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, 15-18May 2001.

[3] J. Closier and al. Results of the lhcb experiment data challenge 2004. In *CHEP'04*, Interlaken, November 2004.

[4] Doar. A better model foor generating test networks. In *IEEE GLOBECOM*, 1996.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[6] Ian Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[7] Garonne, V. and Stokes-Rees, I. and Tsaregorodsev, A. DIRAC: A Scalable Lightweight Architecture for High Throughput Computing. In *Grid 2004, 5th IEEE/ACM International Workshop on Grid Computing*, November 2004.

[8] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 191–202. Springer-Verlag, 2000.

[9] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, and Graham R. Nudd. Optimising static workload allocation in multiclusters. In *IPDPS*, 2004.

[10] L. Kleinrock, editor. *Queueing Systems Volume I : Theory*. John Wiley and Sons, 1975.

[11] Hui Li, David Groep, and Lex Walters. Workload characteristics of a multi-cluster supercomputer. In *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2004.

[12] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[13] D. Lu and P. Dinda. Synthesizing realistic computational grids. In *Proceedings of ACM/IEEE SC 2003*, 2003.

[14] SETI@Home. `http://setiathome.ssl.berkeley.edu/`.

[15] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[16] A. Takefusa. Bricks: A performance evaluation system for scheduling algorithms on the grids. In *JSPS Workshop on Applied Information Technology for Science*, 2001.

[17] Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, and Francine Berman. A study of deadline scheduling for client-server systems on the computational grid. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 406. IEEE Computer Society, 2001.

[18] S. Vadhiyar and J. Dongarra. A metascheduler for the grid. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, 2002.