TSM-SIM: An Evaluation Tool for Grid Workflow Scheduling

Mohamed Amine Belkoura[1]**and Noé Lopez-Benitez**[1]
[1]Department of Computer Science, Texas Tech University, Lubbock, Texas, USA

Abstract— *Implementing efficient workflow job scheduling is a key challenge in grid computing environments. Simulation is an efficient mechanism to evaluate scheduling algorithms and their applicability to various classes of grid applications. Several grid simulation tools exist, and provide a framework for studying grid job execution in conjunction with different scheduling algorithms. However, these simulators are tailored only to independent grid jobs, with limited support for complex grid workflows submission and scheduling.*

This paper presents TSM-SIM, a two-stage metascheduler simulator for grid workflow applications. It supports dynamic grid resource and job simulation, and provides a submission interface for workflow grid applications as a single unit, rather than as a set of grid jobs. We detail the overall architecture TSM-SIM as well as example of its scheduling algorithms. We demonstrate how it can be used to collect performance scheduling data of complex grid workflow benchmarks.

Keywords: metascheduling, task management, middleware, grid, simulator, architecture

1. Introduction

Grid workflows are orchestrated by grid meta-schedulers, matching grid application jobs with available resources, in order to achieve an optimal execution [1]. To satisfy such requirement for complex grid flows, a Two-Stage Metascheduler (TSM) decouples logical task meta-scheduling from physical task/node matchmaking, while achieving an improved overall performance [2]. In order to validate the effectiveness of this scheduling architecture, only a comprehensive and rigorous testing process can produce accurate and meaningful results. Given its inherent complex and dynamic nature, computing grids are hard to evaluate. Setting up grid testbeds that are both realistic and adequately sized is an expensive and time consuming process, and therefore represents a barrier to meta-scheduler algorithm evaluation. We propose the use of grid simulators to measure the efficiency of metascheduling algorithms in a diverse and comprehensive set of scenarios. To evaluate the efficiency of two-stage metascheduling, it is important to run a number of tests, varying different parameters and platform scenarios, with the goal of producing statistically significant quantitative results. However, real-world grid platforms are hard to setup, labor-intensive, and are generally constrained by the available hardware and software infrastructure. To preserve the security and consistence of valuable grid resources, grid administrators tend not to allow users to modify some grid parameters, such us participating nodes, network connections and bandwidth, and some lower level grid middleware and operating system configuration. For all these reasons, a simpler and reproducible approach to evaluate grid application scheduling requires the use of simulators.

2. Background and Related Work

Several solutions were proposed in the realm of grid application scheduling simulation. Bricks simulator [8] is a JAVA simulation framework used to evaluate the performance applications and scheduling algorithms in Grid environments. It consists of a discrete event simulator, a simulated grid computing and data environment, as well as network components. It allows the analysis and comparison of various scheduling algorithms on simulated grid settings, taking into consideration network components effect on the overall performance. SimGrid [6] is another widely used toolkit for the simulation of parallel and grid application scheduling. It supports an out-of-the-box creation of time-shared grid and cluster resources. It also supports varying resource loads statically and dynamically. It also provides an extensibility programming layer for adding or customizing grid jobs and resources creation based on various parameters. Its programming interface provides several mechanisms to implement resource scheduling policies. GridSim [3] is also a popular simulation framework for grid and parallel applications. It supports different resource schedulers, including time-shared and space-shared resources. It contains a network simulation component, used for simulating network topologies, links and switches. it also allows incorporating resource failure into the grid application simulation. OptorSim [9] is a java based grid simulator focusing on data grids. It can simulate grid resources of different storage or computing elements, and allows the testing of data replication strategies. Its scheduling simulation is achieved through a resource broker, which implement scheduling schemes. It treats sites computing or data facility as network nodes and routers. For data replication, it features a replica manager and optimizer that handles advanced data manipulation and management.

While these simulators provide mechanisms for a flexible grid application modeling, they do not support the submission of the whole grid application workflow as an input, with all its data and sequence job dependencies, but only support

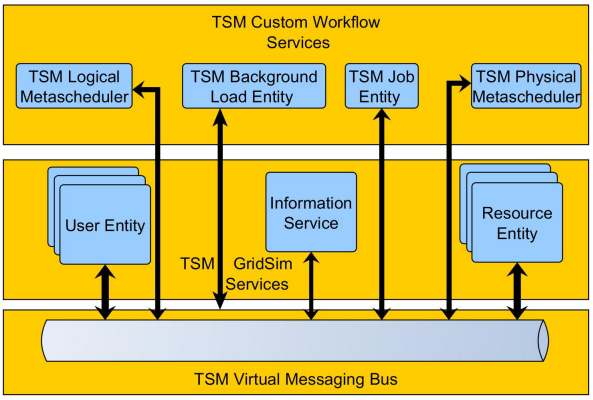

Fig. 1: TSM Simulator System View.

individual jobs submission. The proposed TSM simulator in this paper will extend some of GridSim components, and provide a two stage logical/physical scheduling module based on the TSM architecture.

3. Simulator System Overview

TSM-SIM allows a comprehensive study of the dynamic interaction of multiple grid components, including grid users, resources, networks and various scheduling algorithms. It provides a virtual grid infrastructure that enables grid workflow application experimentation with dynamic meta-scheduling algorithms, supporting controllable, repeatable, and observable experiments. From a system view, TSM-SIM is composed of three main components: TSM Virtual Messaging Bus, TSM GridSim Services, and TSM Custom Workflow Services, as shown in figure 1. It uses an inter-process discrete event based system for communication. Each layer exposes functions for reuse with other services. The following section provides a detailed description of each layer components.

At the core of the TSM simulator is a virtual messaging bus implemented using Simjava framework, inherited from GridSim [5]. Simjava is a inter thread messaging framework that allows sending tagged event from one entity to another within the same java process. Simjava entities are connected to each other using ports and can inter communicate by sending and receiving tagged event objects. A separate thread controls the lifecycle of the entity threads, by synchronizes their execution

4. Simulation Algorithms

In this section, TSM logical and physical metaschedulers algorithms are presented. For the logical metascheduler, details of the Execution Set Algorithm (ESA), the Delayed Execution Set Algorithm (DESA), and the Block Notification Execution Set Algorithm (BNESA) are given. For the physical metascheduler, the ClassAdd Matchmaking Algorithm (CMA), and the Workflow Weight Algorithm (WWA) are outlined.

4.1 Logical Metascheduler Algorithms

TSM scheduling approach consists of decomposing a grid workflow into a set of individual jobs that can be executed in parallel. Any logical algorithm should compose these tasks either dynamically or statically, taking into consideration only job dependencies. The rest of this section presents three logical metascheduler algorithms: the Execution Set Algorithm (ESA), the Delayed Execution Set Algorithm (DESA), and the Block Notification Execution Set Algorithm (BNESA).

4.1.1 Execution Set Algorithm

The Execution Set Algorithm (ESA) accepts as an input the grid application flow in a digraph format (directed graph). It processes the flow composing tasks (graph nodes), data and control flow (edges), and produces a set of task pools, called Execution Sets (E). These sets are submitted to the physical metascheduler in order. None of the task of certain set can be submitted to a physical scheduler unless all the tasks of the preceding set have been submitted. However, the composing tasks of each pool can be submitted in any order. The execution set is updated after execution success notification.

The ESA logic is described by the following algorithm.

ESA Algorithm

$P \leftarrow$Initial flow graph of N nodes.
$E \leftarrow$Current execution set.
$S \leftarrow$Set of nodes submitted, but not executed yet.
$G \leftarrow$Graph of N nodes; G=$\{n_i; i \in [1; N]\}$.
while E $\neq \oslash$ **do**
  Submit S elements for execution.
  if receipt of sucessful execution of n_j **then**
    $E \leftarrow$E - $\{n_j\}$.
    $S \leftarrow$E
    Remove mode n_j and its edges from P
    Update E with additional free nodes of P

4.1.2 Delayed Execution Set Algorithm

The Delayed Execution Set Algorithm (DESA) is a variant of ESA that introduces a delay between the receipt of the first notification, and the submission of the next execution set. A delay allows collecting more individual grid jobs into individual execution sets. This reduces the number of submissions and execution notifications, but generates bigger execution sets. DESA is analyzed with various delay times, in order to study its impact on grid utilization, and total grid application execution time.

4.1.3 Block Notification Execution Set Algorithm

The Block Notification Execution Set Algorithm (BNESA) is a second variant of ESA, where the next algorithm execution set will not be updated immediately after the first successful job notification. Instead the algorithm waits for k numbers of notifications, where k is directly correlated with the size of the execution set E. As for DESA, a delay will allow for potential addition execution notifications, therefore a bigger execution set. This variant is studied with various values of k, in order to analyze its impact on grid utilization and total grid application execution time.

4.2 Physical Metascheduler Algorithms

Different grid job/grid resource algorithms are used in grid environments. They can be classified into three main categories: time-shared, space-shared and backfill algorithms. Time-shared grid scheduling algorithm allocates grid resources in a round robin scheme, and exclusively allocated a grid resource to a grid job until it is completed. Space shared grid scheduling algorithm allocates grid resources in a First Come First Serve (FCFS) and executes more than one processing element (PE) to a grid job. Backfill algorithms attempts to reorder jobs queued to be executed, by moving small jobs ahead of big ones in the scheduling queue. The job prioritization is done to fill in holes in the schedule, without delaying the first job in the queue. Two variants of this class of algorithms exist. The first is called aggressive backfilling, where short jobs will automatically priority over long jobs. The second is called conservative backfilling, where the acceleration of short jobs happens only of such reorder does not delay any job in the schedule queues.

4.2.1 Condor ClassAd Algorithm

TSM physical metascheduler implements *Condor ClassAd Algorithm* (CCA), the standard matchmaking alogorithm used by Condor [10] . Each executing node in the grid advertises its resource ClassAd, while each workflow grid job is defined by its processing requirement within its job ClassAd. For each execution set received from the logical metascheduler, a single round of matchmaking algorithm is made, based on the constraints defined by both the resource and job ClassAds. If a requirement of a job is not met during the matchmaking process, it is delayed until the current execution set is refreshed after the next logical execution set is received. However, if a grid resource is a positive match for a grid job, such matching is selected, and no other possible matchmaking combination is evaluated. In our implementation of CCA, only the requirement part of the ClassAd is considered. The optional rank attributes are not used in our matchmaking process.

4.2.2 Workflow Weight Algorithm

The second TSM physical metascheduler prototyped is called *Workflow Weight Algorithm* (WWA). It is a time shared, first-come-first-served class algorithm that captures the instant load of each grid resource using an auction style election process. Each grid resource ($R_j 1 < i < m$, m is the number of resources) is characterized by its number of machines Rm_i($Rm_i = 1$ for a non-cluster resource), the number of its processing units Rpu_i, its processing power Rpp_i in Million Instructions Per Seconds (MIPS), its available memory $Rmem_i$, and its network connection speed Rn_i. Each grid job/task ($T_j 1 < j < n$, n is the number of tasks) has a strict number of processor units Tpu_j and memory requirement that need to be satisfied at a single grid resource, in order to be considered in the match-making. Each grid job will advertise its computing power need Tpp_j, its memory requirement $Tmem_j$, its total data input size Tin_j, its total output size $Tout_j$, its height in the workflow tree Thi_j, and its offspring count $Toff_j$.

The algorithms works as follow: In the physical metascheduler, a discrete scheduling interval $\nabla\tau$ will be defined (for example a 10 second interval). At each interval beginning, the metascheduler calculates a scalar value called grid task weight *Tweight*$_j$. This value provides a quantifying value of all the computing characteristics of a grid job/task, and is defined as follows:

$$Tweight_j = C_T \times Tpu_j \times Tpp_j \times Tmem_j \times Tin_j \times Tout_j \times Thi_j \times Toff_j, \;\; 1 < j < n \tag{1}$$

where C_T is a constant at each scheduling iteration. Simultaneously, a similar weight, called the Resource Weight *Rweight*$_i$, is calculated. The logical scheduler will request from each grid resource site its dynamic computing data. Only grid resources that are free submit their data, indicating that they are willing to participate in the current scheduling round. The metascheduler will then calculate the Resource Weight *Rweight*$_i$ defined as follows:

$$Rweight_i = C_R \times Rmi_i \times Rpu_i \times Rpp_i \times Rmem_i \times Rn_i \quad 1 < i < m \tag{2}$$

The next step of the algorithm is to sort all the values of Tweight and Rweight in a descending order. A height Rweight value indicates a fast grid resource, while a high Tweight value indicates a demanding grid job/task. The algorithm assigns the grid job/task of highest Tweight value to the grid resource of the highest Rweight value, with the condition that equation 3 satisfied:

$$Tpu_j \leq Rpu_i \text{ and } Tmem_j \leq Rmem_i \quad (3)$$
$$1 < j < n \; and \; 1 < i < m$$

Note that the number of grid tasks “n" is generally different than the number of grid resources “m" (being equal is only one special case). In case the case of n<m, only the available fast grid resources of the grid are being used. In the case of n>m, a resource starvation is happening, and only a portion of the execution set is actually assigned a grid resource. Grid tasks that are not scheduled will be part of the next scheduling round.

5. Grid Benchmarks

NAS Grid Benchmarks (NGB) were used to test the TSM simulator. NGB is a benchmark suite designed by NASA, based on the NAS Parallel Benchmarks (NPB) [11]. The suite contains different classes of workflow application, and thus helps measuring the capability of a grid infrastructure to execute distributed, communicating processes while testing its functionality and efficiency. NGB benchmarks are defined as data flow graphs, with nodes and arcs representing computations and communications respectively. NGB benchmarks are used to measure each node execution time, as well as the data transfer capabilities of the communication network, particularly latency and bandwidth. An instance of NGB benchmark grid flow is a collection of six types of computing programs. They are called Block Traditional solver (BT), Scalar Pentadiagonal solver (SP), Lower-Upper symmetric solver (LU), Multigrid solver (MG), fast Fourier Transform solver (FT), and Mesh Filter solver (MF). Each instance of these programs is characterized by a class, which describes the size of its input data. These programs different classes are S, W, A, B, C. In our experiments, we considered only S, A, B classes. Every benchmark program code (BT, SP, LU, MG, or FT) is specified by class (mesh size), number of iterations, source(s) of the input data, and consumer(s) of solution values. The DFG consists of nodes connected by directed arcs. It is constructed such that there is a directed path from any node to the sink node of the graph. All of NPB's mesh based problems are defined on the three-dimensional unit cube. However, even within the same problem class (S, W, A, B, or C) there are different mesh sizes for the different benchmark codes.

Table 1 gives the problem size and the memory requirement for every computing program used in the NGB benchmark used in [12].

5.1 Class of NGB Benchmarks

NGB benchmarks consist of four families of problems: Embarrassingly Distributed (ED), Helical Chain (HC), Visualization Pipeline (VP), and Mixed Bag (MB). These benchmarks are described in the rest of this section.

Program	Class	Problem Size	Memory requirement (MW)
SP	S	12^3	0.2
	A	64^3	6
	B	102^3	22
BT	S	12^3	0.3
	A	64^3	24
	B	102^3	96
LU	S	12^3	0.3
	A	64^3	30
	B	102^3	122
MG	S	32^3	0.1
	A	256^3	57
	B	256^3	59
FT	S	64^3	2
	A	$256^2 \times 128$	59
	B	$256^2 \times 512$	162

Table 1: NGB programs size and memory requirement

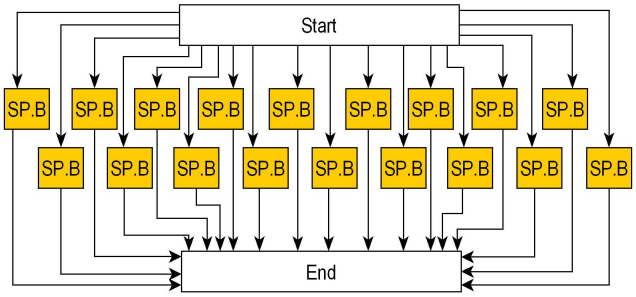

Fig. 2: ED, class B (18x1) grid flow.

5.1.1 Embarrassingly Distributed

The Embarrassingly Distributed (ED) benchmark represents a class of grid applications called parameters studies, where the same basic program is executed multiple times, and each time with a different input data. This class of benchmark models applications that can be obviously divided into a number of independent tasks. The application tasks are executed independently, with different inputs. Figure 2 shows the b-class ED benchmark grid workflow used in our simulations.

5.1.2 Helical Chain

Helical Chain (HC) benchmark models grid application with long chains of repeating programs, such as a set of flow computations executed in order. It consists of a sequence of jobs that model long running simulations that can be divided into different tasks. Each job in the sequence uses the computed solution of its predecessor to initialize. Figure 3 shows an example of a b-class HC benchmark grid workflow.

5.1.3 Visualization Pipeline

Visualization Pipeline (VP) benchmark models grid workflow application composed of multiple chains of compound processes. It represents a chain of grid jobs, but with limited parallelism. It models grid applications where the last itera-

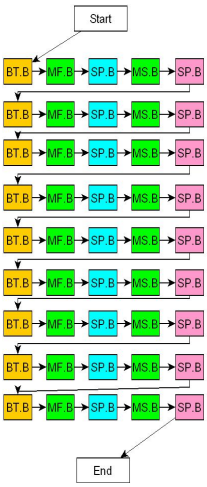

Fig. 3: HC, class B (5x9) grid flow.

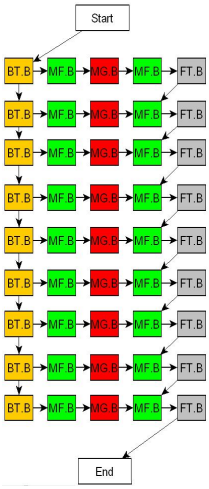

Fig. 4: VP, class B (5x9) grid flow.

tion step is a visualization/analysis task. Figure 4 illustrate an example of a b-class VP benchmark grid workflow.

5.1.4 Mixed Bags

Mixed Bag (MB) benchmark models grid applications composed of post-processing, computation and visualization computing tasks, but with inter asymmetric communication. It also features different tasks that require both different data and computing power. It introduces double and triple dependencies, where some jobs have two or three parent tasks. It constitutes the most complex benchmark in the NGB suite, and thus making it hard for any scheduler to schedule its tasks efficiently. Figure 5 shows an example of a b-class MB benchmark grid workflow.

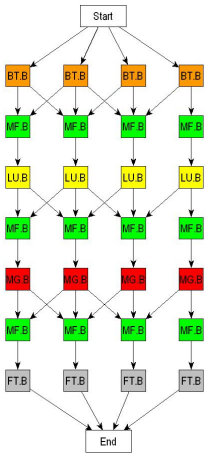

Fig. 5: MB, class B (5x9) grid flow.

Cluster Name	Cluster Name	Node Name	Memory (GB)	Computing Power (MFLOPS)	Network Speed (GB/S)
Main TTU Campus	Cluster 1	Compute1-1	12	9320	10
		Compute1-2	12	9320	10
	Cluster 2	Compute3-1	4	6400	10
		Compute3-2	4	6400	10
	Cluster 3	Compute8-1	64	10400	10
	Cluster 4	Compute6-9	4	6400	10
		Compute6-10	4	6400	10
	Cluster 5	Compute6-1	12	9320	10
		Compute6-2	12	9320	10
TTU Reese Campus	Cluster 6	Compute10-1	4	12000	10
		Compute10-2	4	12000	10
	Cluster 7	Compute11-17	4	9320	10
		Compute11-18	4	9320	10

Table 2: Resource Properties of the Grid Testbed

6. Simulation Environment Setup

In our experiments, we simulated a subset of Texas Tech Hrothgar and Antaeus clusters [13] as part of a computing grid. The grid modeled in our simulations contains 13 resources, spread both among the main and satellite Texas Tech campuses. We modeled each of its grid resources with a total number of processing elements (PEs) characterized by their MIPS rating (Million Instructions per Second) and their internal memory capacity. We also model the network connecting all the grid computing elements, by specifying, the network layout, the number of routers, and the network link properties such as bandwidth in bits/second and Maximum Transmission Unit (MTU) in bits.

Table 2 shows the grid test bed properties simulated, while figure 6 details the network topology of the simulated grid environment.

6.1 Experimental Methodology

A set of each NGB benchmark grid application is generated, and submitted to the TSM simulator, under the different benchmark type (ED, HC, VP and MB), class (S, A, and

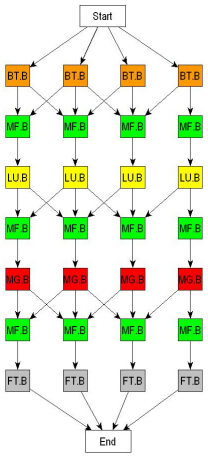

Fig. 6: Network Topology of the Grid Test Bed.

Load Pattern	normal distribution size(MB)		Poisson distribution inter-arrival time (s)	
	Min	Max	Min	Max
Medium	2,5	5	10	50
High	10	20	10	50

Table 3: Background Traffic Generator Pattern Parameters

B), background load (none, medium, high). Both Execution Set Algorithm (ESA) and Delayed Execution Set Algorithm (DESA) for the logical metascheduler and Workflow Weight Algorithm (WWA) for the physical metascheduler were implemented. A background traffic generator is used to simulate a non-exclusive access schema to a grid. The background traffic generator was configured with two different load patterns: a medium and a high load pattern, which are generated based on the size of background data, the job size, and the inter-arrival time. The background load data size follows a normal distribution of a minimum size of 2.5 KB and a maximum of 5 KB. Its arrival time follows a Poisson distribution, with an inter arrival times varying from 10 to 50 seconds. The traffic generator is bound to each resource, so the background traffic and load hits all resources and its network route, starting from the TSM simulator. Table 3 shows the values for each background traffic generator pattern parameters.

6.1.1 Performance Metrics of TSM Algorithms

Various metrics were defined and captured during each simulation execution. The overall performance of a grid workflow application can be measured by the time it takes to finish its total execution, which starts with the time it is submitted to the TSM metascheduler and finishes when the last composing job executes successfully, and its output is received by the TSM metascheduler. We refer to this time as Grid Application Execution Time (T_{GAET}). This execution time takes into consideration the execution of the grid application composing tasks, as well as the network time consumed to transfer input and output files needed by each composing task. Because a grid is a parallel execution environment, T_{GAET} is not the sum of each task execution time, and each file transfer time. As more than a task can execute at the same time, several execution and transfer tasks overlap. On the other hand, contention over grid resources and network connection introduces additional delays counted toward T_{GAET}. In addition, the use of the TSM algorithm introduces another meta-scheduling computing time. As a result, the total workflow execution time consists mainly of three components: a task execution component T_{EX}, a data transfer time component T_{DT}, a meta-scheduling component T_{TSM}, and an idle time component T_{IDLE}, spent either waiting for resources to be available, or when a job is queued at the local grid machine scheduler. Therefore, T_{GAET} calculation formula is obtained as follows.

$$T_{GAET} = T_{EX} + T_{DT} + T_{TSM} + T_{IDLE} \quad (4)$$

T_{EX} is the time spent running the task program on a grid resource, and does not count network time. T_{DT} is the total time use to transfer data in and out of grid resources. T_{TSM} is the total meta-scheduling time taken by both the logical and physical metascheduler to allocate resources to grid jobs will depend on how many times it execute the matchmaking algorithm. T_{IDLE} is the time slot not used for the three main active times is considered idle or unused. We also define the Total Grid Time T_{TGT}, which constitute the total time per grid resource that was spent executing the grid application. It is obtained as follows:

$$T_{TGT} = T_{GAET} \times N_r \quad (5)$$

Where N_r is the number of available grid resources. In our experimental simulated grid environment, $N_r = 13$.

7. Experiment Results

The purpose of these experiments is to study the effect of varying both the background load and the scheduler variant on the performance of the scheduling policies, and show how grid workflow applications benefit from the two-stage scheduling in real workflow situations. It also showcases the value of TSM-SIM producing experimental results for various grid scheduling and load conditions. We first present an analysis of background load on grid workflow scheduling, where we test the combination of ESA/WWA algorithms. Second, we analyze the impact of delayed submission, using the DESA/WWA algorithm combination.

7.1 Background Load Effect

We first consider the effect of background load on scheduling different class of workflow grid applications. The effect of background load effect on the total time is shown in figures 7 to 10. As the load increases, the total time increases, especially for the pure parallel flow (ED), and the

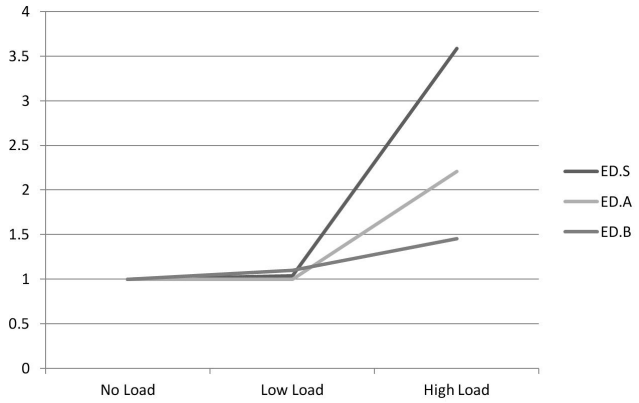

Fig. 7: Load Effect on ED Class Workflow.

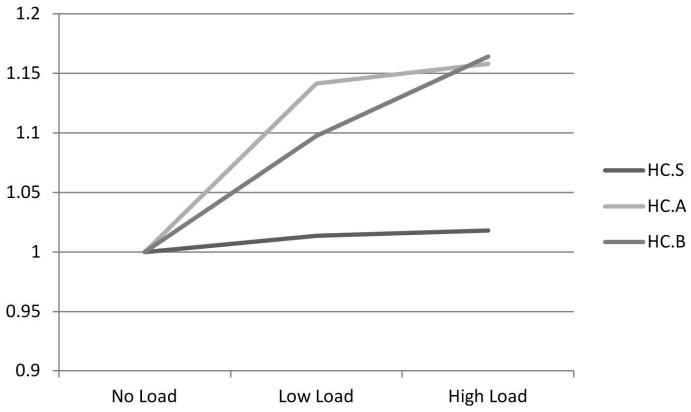

Fig. 8: Load Effect on HC Class Workflow.

pure sequential flow (HC). For the case of (ED), only high background load effect the response time, while it only takes some low background load to slow the workflow execution in case of HC. Also, the effect of high load on the most complex (ED) class workflow is more than 3 times the effect of the type of load on HC class workflow. Note also that slowdown due high load on HC class workflows is more influenced by the size of the data, than by the complexity of the workflow.

For a more general type workflow, such as class VP and MB, the background load have less effect than the case of ED and HC, with a maximum of 1.1 ratios for VP, and of 1.5 for MB. The VP figure (figure 9) shows an insignificant load effect on the total workflow execution time, with no more that 0.1 increase ratio. This means a close to optimal experimental utilization of grid resources. However, the higher level of parallelism in a workflow (case of MB), the more significant is the effect of background load, especially in when it is high. In fact, for the case of MB benchmark, which is the most complex benchmark, the overall slowdown aproaches 50% under sustainable grid load, especially in the case of the MB.B class benchmark.

As a conclusion, we can state that experimental tests using TSM-SIM show that the effect of background grid loads have a higher impact of grid workflows with high level of parallelism (ED and MB). This can be correlated to the average execution set size. In fact, a high level of parallelism in a grid workflow causes the TSM logical metascheduler to generate bigger execution sets. The jobs composing these execution sets are more penalized by the background load, because they also compete with each other for fast resources.

This peer competition effect, while it can also effect non-related grid jobs, significantly impacts grid workflow applications more than isolated grid jobs. The background load effect in case of grid workflows is higher, because its impact on scheduling and execution is compounded.

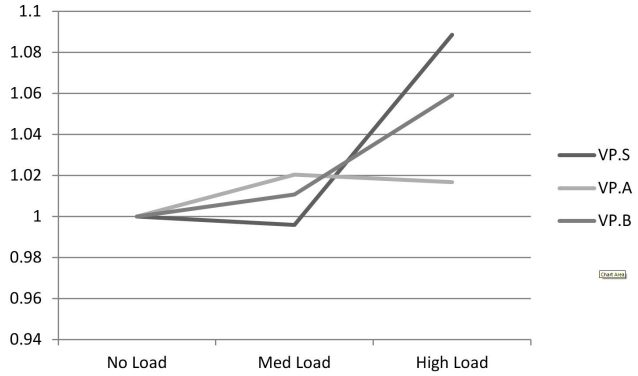

Fig. 9: Load Effect on VP Class Workflow.

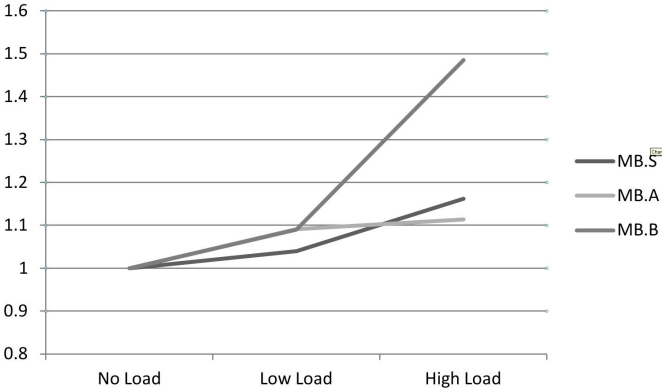

Fig. 10: Load Effect on MB Class Workflow.

7.2 Submission Delay Effect

In this experiment, we test the difference between the ESA/WWA and DESA/WWA, and study the effect of the introduction of a delay in submitting execution sets by the logical meta-scheduler. In a real grid environment, the motivation of such delay is to allow other grid resources to become available, so that a better choice of grid resources is possible. The intention is to wait for potential powerful grid resources to join the pool of available grid resources, which can be beneficial for the overall grid execution. We want to measure if the time wasted waiting for such resources can be easily made up by using a powerful grid resources. The goal of this simulation run is to experimentally study when such strategy is beneficial for grid workflow scheduling, and identify application and environment properties that impact this scheduling strategy. We simulate the submission delay variant (TSM-SDV) in TSM-SIM, by keeping a constant delay of 10 seconds, while varying the background load on the grid infrastructure (grid network and resources). We run 10 simulation of each kind, and measured the average simulation time. We tested this scheduling variant for each grid NGB benchmark (with the S, A, and B complexity classes).

Figures 11 to 14 show the summary of these simulation runs. The Y axis shows the improvement rate $\text{RATE}_{TSM-SDV}$ that DESA contributes to the total workflow execution time compared to ESA. $\text{RATE}_{TSM-SDV}$ is calculated using the following equation:

$$RATE_{TSM-SDV} = Tnd_{GAET}/Tnd_{GAET} \qquad (6)$$

with Tnd_{GAET} is the experiment total workflow execution time in case of no submission delay, and Tnd_{GAET} the same time with submission delay.

The common observation is that submission-delay negatively impacts the grid workflow execution time. In most of the cases, a 50% performance hit is observed. (ED) benchmarks experience the worst performance. The impact can be as much as 500% (rate of 0.2) for the simple class S under no background load. The impact is less visible in case of high than low background load. We can explain this by the 100% parallelism of ED applications. Delaying the submission of the execution set, which contains most of the grid workflow jobs in case of ED benchmarks, gives the opportunity to background load jobs to use fast grid resources. Thus, the penalty of any delay is greater than any benefit that might be achieved. (HC) benchmarks suffer similar negative impact. The performance hit, however, is about constant, varying from a 0.35 to 0.6 factor depending on the load. (MB) benchmarks, the most complex type amount tested benchmarks, do record a similar performance hit in the range of 0.25 to 0.75. The only difference with the impact is amplified by heavy load in a significant proportion. The response time for (VP) type benchmarks seems to be different from other benchmarks. While the behavior seems to be similar than other benchmarks in case of no or little background load, the total grid application performance seems to be un-affected by the submission delay variant. In fact, in case of no load, a slight 10% improvement is recorded, making it the only case where the submission delay benefits the overall grid workflow performance.

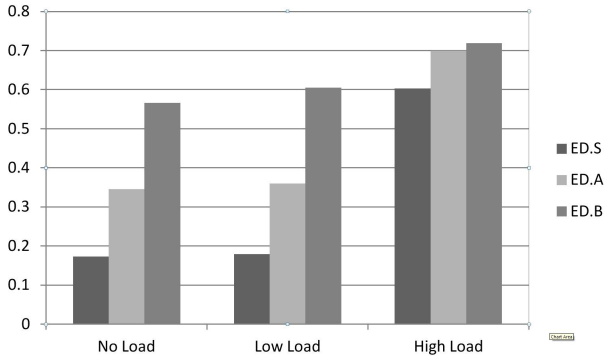

Fig. 11: Submission Delay Effect on ED Class Workflow.

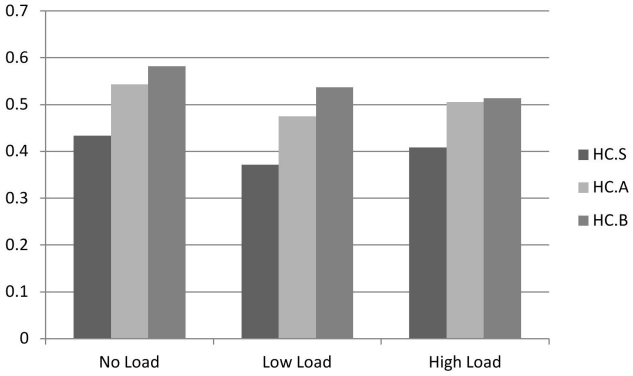

Fig. 12: Submission Delay Effect on HC Class Workflow.

8. Conclusions and Future Work

This paper outlined the details of TSM-SIM, a two-stage grid metascheduling simulator aimed at grid workflow applications. It is primarily intended to test the TSM architecture on a simulated environment, by building on existing GridSim services to configure two-stage scheduling services. We have demonstrated how it can be used to evaluate grid workflow scheduling algorithms using NAS Grid benchmarks.

For future work, TSM-SIM will be used to analyze the performance of other logical and physical scheduling algorithms, using grid workflows. We also intend to build extensibility modules to support high level grid schedulers such as GridWay [14].

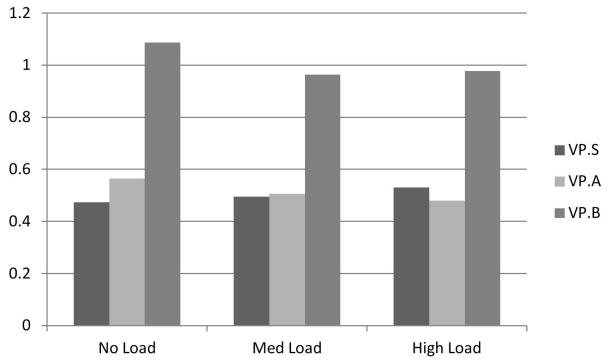

Fig. 13: Submission Delay Effect on VP Class Workflow.

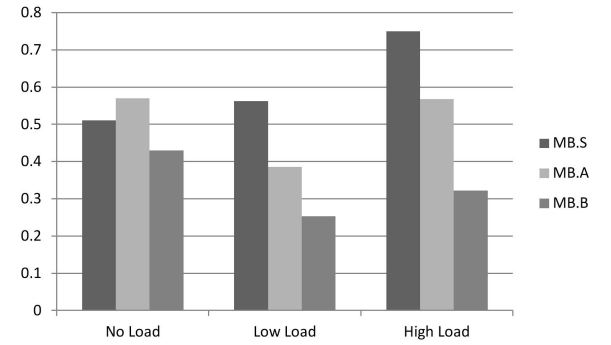

Fig. 14: Submission Delay Effect on MB Class Workflow.

References

[1] M. Amine Belkoura and N. Lopez Benitez, *Two-Stage Metascheduling for Computational Grids*, World Congress in Computer Science, Computer Engineering, and Applied Computing, 2009.

[2] M. Amine Belkoura and N. Lopez Benitez, *TSM-SIM: A Two-Stage Grid Metascheduler Simulator*, International Journal of Grid Computing Applications (IJGCA), 2(4), 11 - 26, 2012.

[3] R. Buyya and M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, Journal of Concurrency and Computation: Practice and Experience (CCPE), 2002.

[4] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic and Rajkumar Buyya, *A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim* , Concurrency and Computation: Practice and Experience (CCPE), Online ISSN: 1532-0634, Printed ISSN: 1532-0626, 20(13): 1591-1609, Wiley Press, New York, USA, 2008.

[5] Agustin Caminero, Anthony Sulistio, Blanca Caminero, Carmen Carrion and Rajkumar Buyya,*Extending GridSim with an Architecture for Failure Detection* , Proc. of the 13th International Conference on Parallel and Distributed Systems (ICPADS 2007), Dec. 5-7, 2007, Hsinchu, Taiwan.

[6] S. De Munck, K. Vanmechelen and J. Broeckhove, *Improving The Scalability of SimGrid Using Dynamic Routing*,Proceedings of ICCS, 2009.

[7] Fred Howell and Ross McNab, *Simjava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling*, in Proc. of First International Conference on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation, 1998.

[8] A. Takefusa, K. Aida, S. Matsuoka, *Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms*, Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8), 1999.

[9] William H. Bell and David G. Cameron and Luigi Capozza and A. Paul Millar and Kurt Stockinger and Floriano Zini, *OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies*, International Journal of High Performance Computing Applications, 2003.

[10] M. Litzkow and M. Livny, *Experience with the Condor Distributed Batch System*, Proceedings. of IEEEWorkshop on Experimental Distributed Systems , 1990.

[11] Michael A. Frumkin and Rob F. Van der Wijngaart, *NAS Grid Benchmarks: A Tool for Grid Space Exploration*, HPDC, 2001.

[12] Rob F. Van Der Wijngaart and Michael Frumkin, *NAS Grid Benchmarks Version 1.0*, 2002.

[13] Anonymous, *HPCC OSG Cluster Grid*, available at http://antaeus.hpcc.ttu.edu/wordpress/.

[14] Huedo, Eduardo and Montero, Ruben S. and Llorente, Ignacio M. *A Framework for Adaptive Execution in Grids*, SoftwareÚPractice Experience, Volume 34 Issue 7, 2004.