TSM-SIM: A TWO-STAGE GRID METASCHEDULER SIMULATOR

Mohamed Amine Belkoura[1] and Noe Lopez-Benitez [2]

[1]Department of Computer Science, Texas Tech University
[1]amine.belkoura@ttu.edu, [2]noe.lopez-benitez@ttu.edu

ABSTRACT

Computing grids must deliver non-trivial qualities of service, in terms of response time, security, reliability, and coordination of various grid resources. Grid meta-schedulers play a crucial role into the execution of applications, because they match grid application jobs with available grid resources. However, a typical grid application is not an isolated single grid job, but a composition of multiple grid jobs. Existing grid schedulers provide match making algorithms tailored only for independent grid jobs, with limited support for complex workflow submission and management.

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. The proposed simulator simplifies complex grid workflow applications scheduling, by implementing two-staged metascheduler architecture. It also allows the study of performance evaluation in a repeatable and controlled manner.

KEYWORDS

Grid, grid computing, simulation, workflow, scheduling.

1. INTRODUCTION

Grid computing has evolved into a novel solution to solve complex computational and data processing problems [1]. The driving force for grid model adoption, especially in the scientific research community, is the possibility to harness virtual computation and data infrastructures, to satisfy unconventional application requirements. A typical scientific grid application is not an isolated single grid task, but a composition of multiple grid jobs and services that require coordinated scheduling and execution. When submitted for execution on a grid, application composing jobs are orchestrated by grid meta-schedulers, matching grid application jobs with available resources, in order to achieve optimal execution. To satisfy the scheduling need of complex grid applications, a two-stage metascheduling architecture as proposed in [2] decouples logical task meta-scheduling from physical task/node matchmaking, while achieving better overall performance. A two-stage grid metascheduler (TSM) relies on a workflow engine, referred to as a logical metascheduler, which identifies composing flow tasks that can be executed in parallel, executes data manipulation and logic associated with job transitions, and selectively submits grid task batches (rather than a complete flow) to a physical metascheduler. The physical metascheduler associates submitted tasks with grid nodes, manages their execution and provides feedback on execution status to the logical metascheduler. The logical/physical metascheduler

DOI: 10.5121/ijgca.2011.2402

interface implements Web Services protocols, and provides greater flexibility regarding the choice of the grid and metascheduling implementation.

The grid simulation scheme presented in this paper seeks to validate the effectiveness of the TSM architecture, through a comprehensive and rigorous testing process. 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 represent a barrier to meta-scheduler algorithm evaluation. Thus, through simulation the efficiency of the TSM architecture is measured by testing targeted metascheduling algorithms in diverse and comprehensive set of scenarios.

The rest of this paper illustrates is organized as follows. Section 3 provides an overview of the TSM simulator system design. Implementation of the logical metascheduler is introduced in Section 4. Following that, Section 5 presents the details physical scheduler, and network modelling. Section 6 shows case of a simulation example using TSM-SIM. We then outline the sequence diagram of a TSM simulation flow, and how each of TSM components interacts with each other in section 4. In section 5, we present both the TSM logical and physical meta-scheduler algorithms. Finally, section 6 illustrates an example grid workflow execution on a modelled grid environment.

2. BACKGROUND AND RELATED WORK

In order 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. In order 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, bandwidth, some lower level grid middleware, and operating system configurations. For all these reasons, a simpler and reproducible approach to evaluate grid application scheduling requires the use of simulators.

Several solutions have been proposed for grid application scheduling simulation. Bricks simulator [11] is a JAVA simulation framework used to evaluate the performance of 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 the effect of network components on the overall performance. However, as it is tailored to support only individual jobs submission, it does not allow grid application workflow as an input, with all its data and sequence job dependencies.

SimGrid [13] is a widely used toolkit for the simulation of parallel and grid application scheduling. It supports out-of-the-box the 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 policy to be simulated. However, it suffers from some inherent limitations, such as the lack of time-shared resource modelling, and the difficulty to simulate background load needed to simulate real grid environments.

GridSim [5] 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 grid application simulation. Like previous solutions, it does not offer a submission mechanism for entire grid application flows that manages dependencies automatically. However, the proposed TSM simulator in this paper will extend some of GridSim components to implement these missing functionalities.

OptorSim [12] 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 computing and data facility sites network nodes and routers. For data replication, it features a replica manager and optimizer that handle advanced data manipulation and management. Since it is a data grid oriented simulation, it lacks advanced resource scheduling, and does not offer support for grid workflow application execution.

Virtual Grid Simulator (ViGs) [10] is a grid environment simulator that analyses performance and scalability of grid applications. It differs from other tools by supporting simulation of real applications. It can model different grid environment components, such as computing resources and networking elements. However, it accepts only single grid job submission, and thus cannot execute a grid application workflow and schedule its composing tasks accordingly.

The motivation for TSM-SIM is the lack of some or all the following features in existing grid simulators:

- **Support for grid workflow application submission as a single unit**: A grid application is generally written as a workflow of atomic jobs, where dependencies can be a complicated mix of data and control flow. None of the workflow grid application elements (programs, data, and control flow) is dependent on a physical resource on the executing grid environment. Discovery of grid resources (data sets, computing nodes, and network links) is often done by querying various grid catalogues. As a consequence, the workflow execution time is challenging to predict, and cannot be simply just the sum of times spent executing all its tasks.

- **Support for real time scheduling**: In a workflow of tasks, critical tasks are those whose execution should have their earliest start times, in order to achieve the best workflow execution time. The sum of the execution times of critical tasks, their data input and output network transfer time, and their scheduling time, is the time spent for workflow task execution. In reality, this gets complicated as the workflow execution progresses. This critical path is changing, because the availability of grid resources, including computing nodes, network connections, and external load changes as the execution progresses. This particular grid workflow scheduling requirement is addressed by new adaptive scheduling approaches, where considerable consideration is given to both the instantaneous status of grid elements, as well as the critical path execution of grid flows [4]. The TSM simulator is designed to analyze scheduling workflow applications, and collect grid performance data in response to different grid scheduling algorithms. It models all active elements of a computing grid, including logical and physical metascheduler, grid resources, and network elements. It simulates resource contention caused by network resources and background load when generating grid workflow schedules. It supports dynamic scheduling schemes, where match-making decisions are made at each step, based on the execution status of previous steps.

3. TSM SIMULATOR SYSTEM OVERVIEW

The main objective of the TSM-SIM is to study and characterize workflow scheduling algorithms performance in grid environments. TSM-SIM allows comprehensive study of the dynamic interaction of multiple grid components, including grid users, resources, networks and various

scheduling algorithms. It provides a transparent virtualization and modelling of such key components, allowing the direct study of complex grid workflow, whose internal dynamics are difficult to model accurately. In short, TSM SIM provides a virtual grid infrastructure that enables grid workflow application experimentation with dynamic meta-scheduling algorithms, supporting controllable, repeatable, and observable experiments.

3.1. Simulator Organization

The TSM Simulator software organization consists of three main layers: 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.

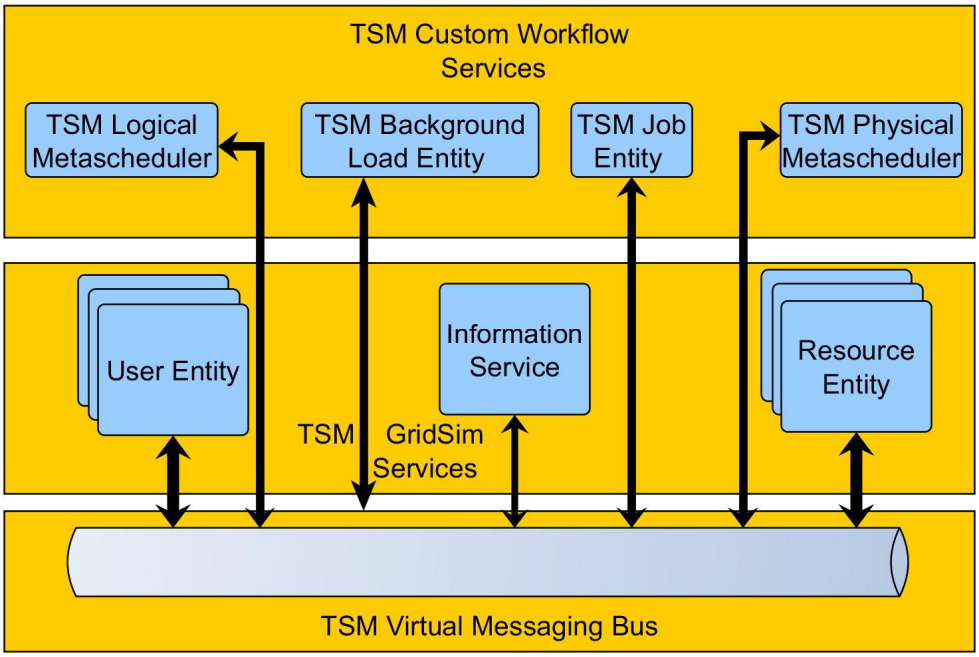

Figure 1: TSM Simulator System View

3.1.1. Virtual Messaging Bus

At the core of the simulator is a virtual messaging bus implemented using the 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 [9].
Figure 2 illustrates the main components of Simjava messaging system. It shows that "entity 1" is sending an event to "entity 3" through the virtual bus. TSM Simulator virtual bus simulates the network and communication links that connects real computing grid components. Simjava component of GridSim was extended with new defined messages and message types [8], in order to accommodate the requirements of TSM logical and physical metascheduler defined later in this paper.

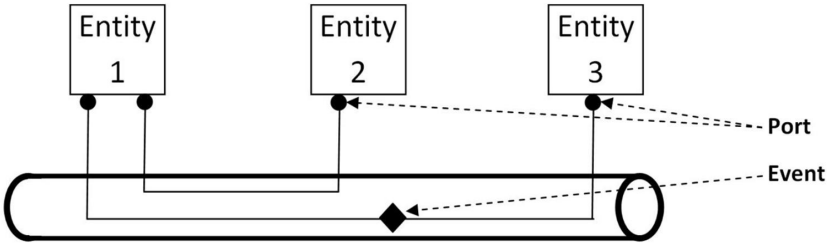

Figure 2: TSM Virtual Messaging Bus

3.1.2. GridSim Services

TSM uses GridSim services [6] to abstract core grid entities. It supports the creation of the main entities for users, brokers, resources, information service, statistics, and network based I/O. The following describes the functionality that each GridSim entity encapsulates.

- User Entity: Each instance of the user entity represents a grid user. The user entity is connected to the virtual grid simulation bus with a network link. Each user entity defines the user grid connection bandwidth, its network delay, and its network link Maximum Network Unit (MTU). For workflow scheduling, the user entity will keep track for the jobs execution set, and will update it as the workflow execution progresses.

- Grid Resource Entity: grid resource entities represent computing and data storage units that perform the computing task. Grid resource entities can simulate any physical resources, including homogeneous clusters, heterogeneous grid resources, or even on a single computer. Each grid resource entity is characterized by its the number of processing elements (CPUs), its processing power in Million Instructions Per Second (MIPS), its internal allocation strategy, its running operating system and architecture. It can also be assigned some dynamic properties, such as current load.

Grid resource will advertise both its static and dynamic properties to the Grid Information Service (GIS), but also on demand if requested by other TSM components. For example, in case of auction based scheduling, the auction broker will request the grid resource dynamic load, in order to assign resources to jobs efficiently.

- Grid Information Service: Grid Information Service is the central repository entity that maintains a set of static and dynamic grid characteristics. It maintains an update copy of all available registered resources, so other entities can query the Grid Information Service for which resources are available at a certain time, and retrieve the instant load of active resources.

3.1.3. Custom Workflow Services

GridSim framework is intended to execute grid application simulation, where jobs are not connected in a workflow. In order to simulate the behaviour of a complex grid workflow processing, and evaluate TSM algorithms, new structures and functionalities were implemented beyond the existing GridSim components. This section describes the functionality implemented within each workflow related component.

- Job Entity: The job entity extends the basic GridSim gridlet entity, which encapsulate a grid single job. It contains a linked list that points to each job parents and children, in order to capture job dependencies. Job entity contains, in addition to standard job attributes such as job

size in Million of Instructions Per Second (MIPS), job input and output data, two job lists for the job parents and children. These dependencies are captured, because it is used by the logical metascheduler to build the execution set at each logical metascheduling iteration.

- Logical Metascheduler Entity: This entity implements the logic and algorithms for the logical metascheduler. It takes a grid application workflow as input, and then builds execution sets to be sent to the physical metascheduler at discrete intervals. Elements are added to the submission set only if its predecessors have been executed successfully. The discrete interval can be configured so that the execution set update is done more or less regularly.

- Physical Metascheduler Entity: The physical metascheduler allocates the grid resources to each execution set components. TSM Physical Metascheduler performs resource discovery and evaluation searches in grid information services for resources that match the resource specification of a workflow task. The resources discovered by this step are those that match the requirements of hardware architecture and software configuration of the workflow task, for example, the CPU architecture, the operating system and version, the total processing units and the software installed on the resource. This initial resource selection step is only preliminary, and does not evaluate the performance a resource can provide for the task execution. It only finds the resources that are able to "execute" the task. Following this initial selection, TSM Physical Metascheduler will apply a predefined algorithm to elect the best matching resource for each grid job. Section5.3 outlines the algorithm proposed part of TSM-SIM.

- Background Load Generator Entity: The background load generator generates external traffic, and submits it to the simulated grid components, the same way real applications do. It generates network traffic, as well as grid resource load. Background traffic data will travel the network elements (routers and nodes) from the user entity to the resource entity and back. In order to simulate real load situations, it uses Poisson and normal distribution algorithms to generate its data size and inter-arrival parameters.

4. SIMULATION USER SCENARIOS

To use the simulator, a user traces the following steps:

1. Define a workflow application, specifying the properties of every composing job. The user should also specify each job set of parents and children.
2. Define a set of physical grid resources, with their computing properties.
3. Define the complete network map: this includes all the network links and routers between all the computing resources, as well as between the resources and the grid user submitting the grid workflow application. The TSM simulator supports various network topologies, and allows the creation of diverse network elements such as routers and network links. The simulator also supports defining link capacity and link latency for network elements.
4. Define logical and physical metascheduler algorithms,
5. Start the simulator, which triggers the grid application submission to the simulation engine.
6. Observe the execution of the application, and collect results and performance data.

Once the simulator starts, it creates one or more grid users. Each grid user will create a separate instance of a logical and physical metascheduler. The simulator creates a single processing thread for each participating element that run independently of other elements. These elements communicate with each other through a discrete event communication bus. For example, for a logical simulator to send a job to the physical simulator, it creates a logical simulation event; populate it with the grid job properties such as its input data and processing requirements. It will

then tag such event to indicate that it is a job submission and intended to be consumed by the physical simulator. It will then place it in the simulation communication bus, pending that the physical simulator entity will retrieve it for consumption. Such single logical bus, with tagged events constitutes the core inter-thread messaging framework that implements a fire-and-forget messaging paradigm. The logical metascheduler obtains the list of grid resources associated with the virtual grid environment from the Grid information service. The information service acts as a central collector to the grid element state, including available grid resources and their properties.

The scheduling sequence of events is outlined in Figure 3. The user entity will build an execution set, based on the grid jobs that have their dependencies resolved. Such execution set is then submitted to the workflow broker, which will check enough resources are available to execute all the submission set. Note that this initial check does not map one-to-one a grid job to a grid resource, but rather verifies that enough resources with the minimum requirement are available at that stage. If the number of resources available are not enough for all the grid jobs, some jobs will be put in the next execution set, and will be submitted again to the workflow broker in the next simulation tick. Otherwise, the set will be marked as ready for logical physical scheduling, and will be submitted to the auction broker. This broker will create a workflow weight scheduler for each execution set, and will initiate an auction. The purpose of the auction is to let all the available resources to "bid" for the execution of this job. The selection process follows the algorithm defined later in section 5.2. Once the winners are selected, each job is submitted to each selected grid resource. Upon completion, the workflow broker is notified, which triggers the user entity to mark such job as executed, and rebuilds a new execution set. When no job is left, the simulation is marked as complete.

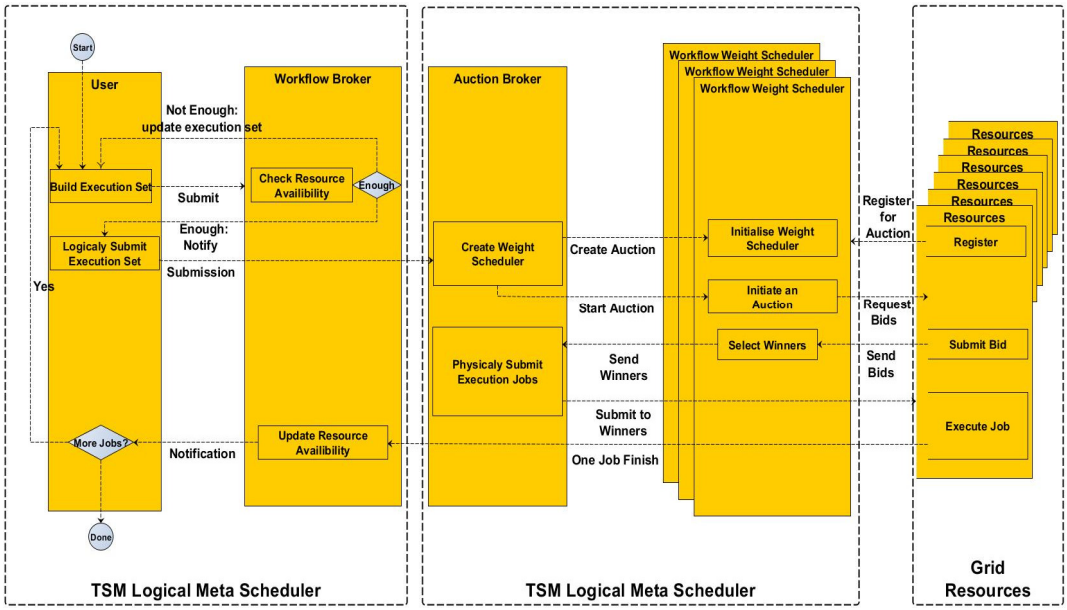

Figure 3: Simulation Sequence Diagram in TSM-SIM

5. SCHEDULING ALGORITHMS

In this section, both logical and physical metaschedulers are presented and different algorithms are presented.

5.1. Logical Metascheduler Algorithms

The simulator accepts the grid application flow as a digraph (directed graph). It processes the flow composing tasks (graph nodes), data and control flow (edges), and produces a set of task pools, called Execution Set (E) that 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 jobs 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 each execution success notification.

The following sections define the basic and modified logical metascheduler algorithms.

5.1.1. Base Logical Algorithm

The base logical scheduling algorithm is defined by the algorithm shown in Figure 4. It outlines the process of building the execution set E.

$P \leftarrow$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

Figure 4: Base TSM Logical Scheduling Algorithm

5.1.2. Logical Algorithm Variants

Two variants of the base TSM algorithms are considered. The goal is to introduce additional tuning factors specific to certain grid applications and test bed layouts.

- Submission Delay Variant (TSM-SDV): this variant introduces of a delay between the receipt of the first notification, and the submission of the next execution set. A delay will allow for potentially more execution notifications, therefore a bigger execution set. This variant will be studied with various delay times, in order to analyze its impact on grid utilization of the grid, and total grid application execution time.

- Block Notification Variant (TSM-BTV): In this TSM, the next algorithm execution set will not be updated immediately after the first successful job notification. Instead the algorithm will wait for a certain k numbers of notification, where k is directly correlated with the size of E. As for the previous variant, a delay will allow for potential addition execution notifications, therefore a bigger execution set. This variant will be studied with various values of k, in order to analyze its impact on grid utilization of the grid, and total grid application execution time.

5.2. 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 purpose of job prioritization is 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 carry higher 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.

For TSM-SIM physical scheduler, we define an algorithm we call *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. The following section describes the auction model WWA implements.

Each grid resource (R_i $1 < i < m$) is characterized by its number of machines Rm_i (where $Rm_i=1$ 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 , $1 < j < n$) defines the 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 process. Each grid job will advertise its computing power needs 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 offspring count corresponds to the number of leaves in the grid workflow tree, while the workflow high corresponds to the longest path from the tree root to its leaves.

The algorithms works as follow: In the physical metascheduler, a discrete scheduling interval $\Delta\tau$ will be defined (for example a 10 second interval). At the beginning of each interval, the metascheduler will calculate a scalar value referred to as grid task weight $Tweight_j$. This is a quantifying value of all the computing characteristics of a grid job/task, and is defined as follows:

$$Tweight_j=C_T \times Tpu_j \times Tpp_j \times Tmem_j \times Tin_j \times Tout_j \times Thi_j \times Toff_j \quad (1)$$

where C_T is a constant at each scheduling iteration.
Simultaneously, a similar weight, referred to as the Resource Weight $Rweight_i$, will be calculated. The logical scheduler will request from each grid resource site to submit some dynamic computing data. Only grid resources that are free can 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 by the following equation:

$$Rweight_i=C_R \times Rmi_i \times Rpu_i \times Rpp_i \times Rmem_i \times Rn_i \quad (2)$$

The next step sorts all *Tweight* and *Rweight* values 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 the following hard requirements of processing units and required memory are satisfied:

$$Tpu_j < Rpu_i$$
$$Tmem_j < Rmem_j$$

Note that the number of grid tasks "*n*" is generally different from the number of grid resources "*m*" (being equal is a special case). In 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.

6. SIMULATION EXAMPLE

To illustrate the simulation process a sample grid workflow application is used. The sample consists of a 10-job grid workflow defined by its graph defined in Figure 5.

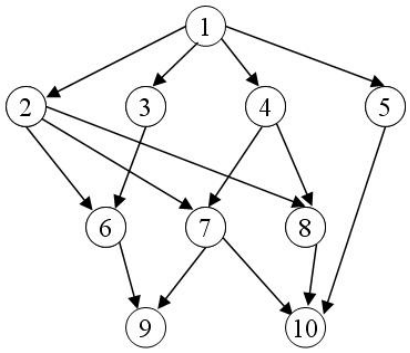

Figure 5: A 10-job workflow Grid Application

6.1. Workflow Application Definition

The simulator accepts a workflow application specification as an ASCII text file. The input definition file has a pre-determined format that contains the total number of workflow jobs, as well as the following grid job properties:

- Job number.
- Job computing requirement in MIPS.
- Job total data input size.
- Job total data output size.
- List of parent jobs.

Figure 6 shows a listing of the grid application workflow definition file given in Figure 5.

```
# Example Grid workflow Application
# Composed of 10 grid jobs
# Comments are appended with a pound (#) sign

# The number of jobs
10

# The list of jobs, with the following syntax
# Job number ! Number of MIPS required ! Total input size in Bytes ! Total output size in Bytes !
#      Operating System ! Machine Hardware ! Not Used ! Not Used! list of parent jobs!

1!721462!8489664!8489664!Solaris!Sun Ultra!0!0!!
2!10!10000!967543!Windows!Intel!0!0!1!
3!62162!44894!87896!Solaris!Sun Ultra!0!0!1!
4!261!13824!13824!Windows!Intel!0!0!1!
5!103!11230!34110!Solaris!Sun Ultra!0!0!1!
6!265!134218!137728!Solaris!Sun Ultra!0!0!2,3!
7!10!262144!262144!Linux!Intel!0!0!2,4!
8!10!11230!13824!Windows!Intel!0!0!2,4!
9!363!23824!23899!Solaris!Sun Ultra!0!0!6,7!
10!650!349810!652710!Solaris!Sun Ultra!0!0!5,7,8!
```

Figure 6: Grid Application Workflow Definition File

6.2. Grid Resource Definition

Similar to the grid workflow application definition, the simulator accepts the grid resource definition as an input file. The grid resources description file is an ASCII text file that contains the following information:

- Resource name.
- Resource link bandwidth in bits/second.
- Resource link propagation delay in seconds.
- Resource link maximum transmission units (MTU) in bits.
- Resource architecture.
- Resource operating system.
- Resource number of machines.
- Resource number of processing units/CPUs.
- Processing unit million instructions per second (MIPS).
- Resource allocation strategy.

Figure 7 shows a listing of the grid resource definition file.

```
# Example Grid workflow Application
# Composed of 10 grid jobs
# Comments are appended with a pound (#) sign

# The number of jobs
10

# The list of jobs, with the following syntax
# Job number ! Number of MIPS required ! Total input size in Bytes ! Total output size in Bytes !
#      Operating System ! Machine Hardware ! Not Used ! Not Used! list of parent jobs!

1!721462!8489664!8489664!Linux!Intel!0!0!!
2!10!10000!967543!Linux!Intel!0!0!1!
3!62162!44894!87896!Linux!Intel!0!0!1!
4!261!13824!13824!Linux!Intel!0!0!1!
5!103!11230!34110!Linux!Intel!0!0!1!
6!265!134218!137728!Linux!Intel!0!0!2,3!
7!10!262144!262144!Linux!Intel!0!0!2,4!
8!10!11230!13824!Linux!Intel!0!0!2,4!
9!363!23824!23899!Linux!Intel!0!0!6,7!
10!650!349810!652710!Linux!Intel!0!0!5,7,8!
```

Figure 7: Grid Resources Definition File

6.3. Network Map Definition

The simulator uses a network map definition file to initialize its network entities and their properties. The input network description file is a text file that lists the number of routers, their speed, as well as all the network links that connects grid resources to routers, as well as inter-router links.

An example of a network map description file is shown in Figure8.

```
# Router information
# Number of routers
2
# Number of inter router links
1
# Number of router/host links
4
# Bandwidth between the user and the router 0
10000000000
# Inter router link information
# source router!target router!link bandwidth in bits/second !
#  Propagarion delay ! Maximum Transmission Unit in bits!
0!1!100000000000!100!512000!

# router-resource link information
# source router!target router!link bandwidth in bits/second !
# First router with Main campus
1!0!10000000000
1!1!10000000000
1!2!10000000000
1!3!10000000000
```

Figure 8: Grid Network Map Definition File

6.4. Simulator Output

The simulator tracks different execution performance values. Each job is tracked from the time it is submitted to the logical metascheduler, to the end of its execution. This includes the duration of its logical scheduling, its physical scheduling, its input data transfer, its execution time, and its output data transfer.

Before each simulation execution, various execution parameters are set. This includes the logical metascheduler variant (none, TSM-SDV, TSM-BTV), the logical metascheduler constants (duration constant for TSM-SDV, correlation constant for TSM-BTV), the background load factor (no load, light load, high load).

Figure 9 shows a summary execution report, listing job execution order, start and end times, as well as the grid resource selected for each job.

```
Starting TSM-SIM simulator...
Initialising...

[User] ============= General Simulation OUTPUT ==========
[User] Gridlet ID   Resource name   Time    Length(MFLOPS)  Execution Start  Execution End
[User]      1       compute-1-1     78.0    721462          607.1            685.1
[User]      5       compute-1-1     12.0    110340          1580.8           1592.8
[User]      2       compute-1-2     15.0    132200          1580.5           1595.5
[User]      3       compute-3-1     26.0    162162          1580.6           1606.6
[User]      4       compute-3-2     34.0    216110          1580.7           1614.7
[User]      6       compute-1-2     3.0     26550           2729.5           2732.5
[User]      8       compute-1-1     15.0    135070          2729.7           2744.7
[User]      7       compute-3-1     19.0    120060          2729.6           2748.6
[User]      9       compute-1-1     4.0     363800          3533.5           3537.5
[User]      10      compute-1-2     18.0    165900          3535.2           3553.2
```

Figure 9: TSM-SIM Execution Report Summary

Figure 10 shows a section of a verbose version of the execution report, listing other key data such as simulation time for data transfer beginning as end, scheduler time per job. The report shows only the output relevant to job 1, 5, and 2, for clarity purposes.

```
[User] Gridlet ID: 1
Time below denotes the simulation time.
Time (sec)       Description Gridlet #1
-----------------------------------------
545.225   Gridlet Submitted to TSM
605.225   Input Data Stage-in
607.526   Execution of Gridlet start on compute-1-1
685.526   Execution of Gridlet ends, data Stage-out starts
687.427   End of data transfer, resource freed.

[User] ------------------------------------
[User] Gridlet ID: 5
Time below denotes the simulation time.
Time (sec)       Description Gridlet #5
-----------------------------------------
1519.805   Gridlet Submitted to TSM
1579.805   Input Data Stage-in
1580.805   Execution of Gridlet start on compute-1-1
1592.805   Execution of Gridlet ends, data Stage-out starts
1593.105   End of data transfer, resource freed

[User] ------------------------------------
[User] Gridlet ID: 2
Time below denotes the simulation time.
Time (sec)       Description Gridlet #2
-----------------------------------------
1519.805   Gridlet Submitted to TSM
1579.805   Input Data Stage-in
1580.505   Execution of Gridlet start on compute-1-2
1595.505   Execution of Gridlet ends, data Stage-out starts
1595.906   End of data transfer, resource freed
```

Figure 10: Per Job TSM-SIM Detailed Execution Report

7. CONCLUSIONS AND FUTURE WORK

This paper outlines 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 build customized two-stage scheduling services. TSM-SIM provides a realistic simulation of grid workflow application in a dynamic grid environment, taking into consideration grid resources load and background network traffic. TSM adaptive framework is well suited to analyse complex grid application workflows and execution environments, such us grids and computing clouds [3].

Future work will focus on identifying grid workflow applications that will benefit from the two-stage metascheduler. With this purpose in mind, we are currently testing TSM-SIM with a set of NAS Grid benchmarks, a grid reference benchmark suite will be used to identify a specific class of grid workflow applications that might perform better under the proposed TSM scheduler. Future simulations will model the Texas Tech University computing grid. The impact of other factors, such as grid load metrics, and scheduling overhead will be explored.

REFERENCES

[1] Foster, I & Kesselman,C (2004) "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann.

[2] Belkoura, M. A & Lopez Benitez, N (2009) "Two-Stage Metascheduling for Computational Grids", World Congress in Computer Science, Computer Engineering, and Applied Computing.

[3] Pankaj Deep Kaur, and Inderveer Chana (2011), "Enhancing Grid Resource Scheduling Algorithms for Cloud Environments", High Performance Architecture and Grid Computing HPAGC 2011, pp 140-144, Heidelberg, Springer

[4] R.P. Prado, J.E. Muñoz Expósito, and S. García-Galán (2011), "Improving Expert Meta-schedulers for Grid Computing through Weighted Rules Evolution", Proceedings of the 9th international conference on Fuzzy logic and applications WILF'11, pp 204-211

[5] Buyya R &Murshed M (2002) "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).

[6] Sulistio, A & Cibej, U & Venugopal, S & Robic,B & Buyya, R (2008) "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.

[7] Caminero, A & Sulistio, A & Caminero, B & Carrion, C & Buyya, R (2007) "Extending GridSim with an Architecture for Failure Detection", Proceeding. of the 13th International Conference on Parallel and Distributed Systems (ICPADS 2007).

[8] Sulistio, A &Poduval, G & Buyya,R & Tham,C (2007) "On Incorporating Differentiated Levels of Network Service into GridSim" , Future Generation Computer Systems (FGCS), ISSN: 0167-739X, Volume 23, Issue 4, May 2007, Pages: 606-615 Elsevier Science, Amsterdam, The Netherlands, May 2007.

[9] Howell, F & McNab, R (1998) "Simjava: a discrete event simulation package for Java with applications in computer systems modelling", Proceeding of First International Conference on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation, Jan 1998.

[10] Thor, A. T. & Za'ruba, G. V. & Levine, D & De, K & Wenaus, T. J (2008) "ViGs: A Grid Simulation and Monitoring Tool for ATLAS", Proceedings of Many-Task Computing on Grids and SuperComputers (MTAGS), ACM/IEEE SuperComputing'08, November, 2008.

[11] Takefusa, A & Aida,K, & Matsuoka, S (1999) "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.

[12] Bell, W. H. & Cameron, D.G & Capozza, L & Millar, A. P. & Stockinger,K & Zini,F (2003) "OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies", International Journal of High Performance Computing Applications, 2003.

[13] Frumkin, M.A & Van der Wijngaart, R. F (2001) "NAS Grid Benchmarks: A Tool for Grid Space Exploration", HPDC, 2001.

[14] Van Der Wijngaart, R.F & Frumkin, M (2002) "NAS Grid Benchmarks Version 1.0".

Authors

Mohamed Amine Belkoura received his Ingénieur d'Etat diploma (equivalent to a bachelor's) from Mohamed V University, Mohamadia's School of Engineering, Morocco and a Master's of Science in Computer Sciences from Texas Tech University, USA. He is currently a PhD candidate at Texas Tech University. His research interests include grid computing, workflow analysis and scheduling.

Noe Lopez-Benitez received a BSEE degree from the University of Guadalajara, MX, a MSEE from the University of Kentucky, and a PhD degree in Computer Engineering from Purdue University. His research interests include Grid/Cloud Computing and Performance/Reliability modelling. His is currently an Associate Professor in the Department of Computer Science at Texas Tech University.