A fully distributed Grid meta scheduling method for non dedicated resources

Haitham Barkallah[1], Mariem Gzara[2], Hanene Ben Abdallah[3]
[1]University of Sfax, Mir@cle laboratory
[2]Higher Institute of Computer Science and Mathematics of Mounastir, Tunisia
[3]King Abdulaziz University, Jeddah, KSA
Haitham.barkallah@gmail.com mariem.gzara@isimsf.rnu.tn hbenabdallah@kau.edu.sa

Abstract—**The purpose of Grid technologies is to support the sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations. As such, jobs scheduling and load balancing functionalities are crucial for best Grid performance and utilization. This paper presents a new meta-scheduling method called TunGrid. It is inspired from the natural phenomenon of heat propagation and thermal equilibrium. TunGrid is based on a Grid polyhedron topology with a spherical like structure. It ensures load balancing through a local neighborhood propagation strategy of overload. Experimental results compared to the RoundRobin and Shortest First algorithms are presented.**

Keywords; Grid computing; meta-scheduling; Load Balincing; polyhedric topology.

I. Introduction

Grid computing [1] has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

There are hundreds of computer Grids around the world for different needs. Grid computing is used by governments and international organizations, military, universities… In fact, Grid computing has enabled projects that would be impossible without massive computing power. Many research groups focus their works on Grid architectures, toolkits, simulation environments, applications…

One of the most important Grid system functionalities is scheduling and load balancing. This work presents a new innovative efficient, scalable, and reliable fully decentralized Grid meta-scheduling method for non-dedicated resources we called it ***TunGrid***. Our method provides parallel job execution and ensures resource load balancing. It method is based on a new Grid topological structure for inspired from the natural phenomenon of heat propagation and thermal equilibrium. This Grid topology is a polyhedron with a spherical like structure. It facilitates the definition of a neighborhood load balancing strategy emulating heat propagation. This paper shows the evaluation of TunGrid using the GSSIM simulator [2].

The reminder of the paper is organized as follows: Section II gives a brief state of the art about Grid meta-scheduling, Section III presents the intuition behind TunGrid. Section IV and V describe, respectively, the ***TunGrid*** topology and load balancing strategy. Section VI is discusses the simulation results. Finally, the paper is concluded with a highlight on some future directions.

II. State of the Art

To achieve the promising potentials of massively distributed resources, effective and efficient scheduling algorithms are fundamentally important. Scheduling problems are encountered in all types of systems where it is necessary to organize and/or distribute the work between multiple entities. In Grid environment, "*Scheduling is a sophisticated decision making that operates at different levels of Grids. Local scheduling is used at the level of clusters, usually to balance load. Global schedulers (Grid schedulers) are used to map user jobs to resources according to their requirements and properties. Higher level schedulers can be used to select brokers or Grids to a specific job.*"[2].

Grid scheduling is also called *super-scheduling, meta-scheduling, scheduling at the Grid level.* Grid resource management systems and schedulers are responsible for the selection and allocation of Grid resources to applications. They handle Grid applications that consist of one or more tasks dynamically and continuously arriving and possibly communicating with one another to collaborate and form a single application.

In the literature, many Grid scheduling algorithms were proposed (*cf.* [3] [4] [5]). They include centralized and decentralized algorithms; Marcin Krystek et al. [6], present a thorough comparison of both approaches of Grid Schduling algorithms.

Some of the proposed scheduling algorithms are static, with and without performance estimate, for independent task scheduling e.g., *minimum execution time*, *minimum completion time*, and *min-min and max-min heuristics* [7]. Others are dynamic meta-scheduling algorithms, e.g. *First-Come-First-Served*, *Earliest-Dead-Line-First Algorithm* [8], *Least-Laxity Algorithm*[9], *Sufferage heuristic* [10].

The proposed algorithms differ in their objectives: Antonella Galizia Alfonso Quarati [11] focused on reducing energy consumption while Wei-Chang Yeh and Shang-Chia Wei [12] tried to minimize the execution cost and increase the profit, Haruna Ahmed Abba and al. [13] considered the job deadline as the prime attribute for job execution and R. Buyya

978-1-4799-3351-8/14/$31.00 ©2014 IEEE

and al. [14] and S. Abrishami and al. [15] proposed solutions to compromise execution cost and deadline respect. In this work, we aim both to decentralize fully the scheduling process in order to eliminate the single failure point and optimize the resource utilization and minimizing execution time.

III. Inspiration Source of TunGrid

The idea behind ***TunGrid*** was inspired from studying the natural phenomenon of heat propagation in physics (*cf.* [2] [3] [5]). Heat is a form of kinetic energy in the vibrations of atoms and molecules that make up matter. The hotter an object is, the more its atoms vibrate and the more heat it contains. The trapped heat in an object is measured by its temperature.

Two bodies in contact tend to equalize their temperatures. Heat spreads from a hot body to a cold body. This is called heat exchange process (or thermal equilibrium). The body having the lowest temperature absorbs a certain amount of heat from the body with the higher temperature. Heat can move from one place to another by conduction, convection and radiation. The absorption rate depends on the difference between the temperatures of the two bodies and the shape of the bodies.

A sphere is a well adapted form to heat spreading [16][17]. Heat spreads from the point of contact in a balanced manner until the total coverage of the spherical surface or total absorption.

Based on this natural phenomenon, in this work, we propose to model the Grid as a polyhedron that is as near as possible to the spherical form. The nodes of the Grid are logically connected to one another. The heat at one point of the sphere corresponds to the jobs queued at the corresponding node of the Grid. The heat spreads to reach the thermal equilibrium; that is, jobs move from more loaded nodes (hotter nodes) to less loaded nodes (cooler nodes) so as to reach a load balance among the Grid nodes.

IV. TunGrid Topology

The topological structure of the Grid is a polyhedron with a spherical like form. Let n be the set of nodes in the Grid. Each node represents a Grid resource. If the number of nodes is a power of 2, then the Grid architecture is a 2D toroid Grid. Each node is connected to its four neighboring nodes, the one at each of its left, right, top and down sides. If this structure is not possible, then we connect nodes so as to obtain a spherical like structure.

The construction of this structure is performed in two steps:

1. connect the nodes to form a flat 2D Grid like structure;
2. Add connections to obtain the final spherical like form.

To clarify this Grid topology, let us start by describing the first "flat view" of the Grid. We construct a 2D Grid composed of p lines and m columns where $m = \left\lfloor \frac{n}{\lfloor\sqrt{n}\rfloor} \right\rfloor$ and $p = \lfloor\sqrt{n}\rfloor$. If $n = m \times p$ then we obtain a matrix of size $p \times m$. Otherwise, we add a partial $(p+1)$ line that contains only $k = n - (m \times p)$ nodes. Figure 1 shows an example of the resulting 2D Grid for $n = 8$. The m and p were chosen in order to obtain a square matrix that will be used to construct the polyhedral form of the Grid.

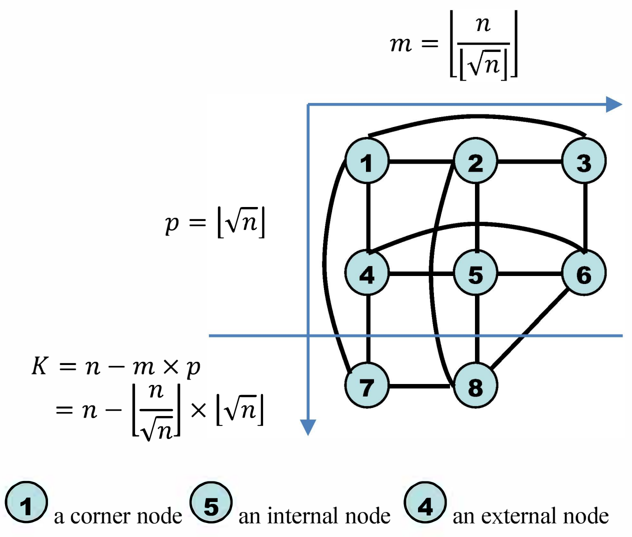
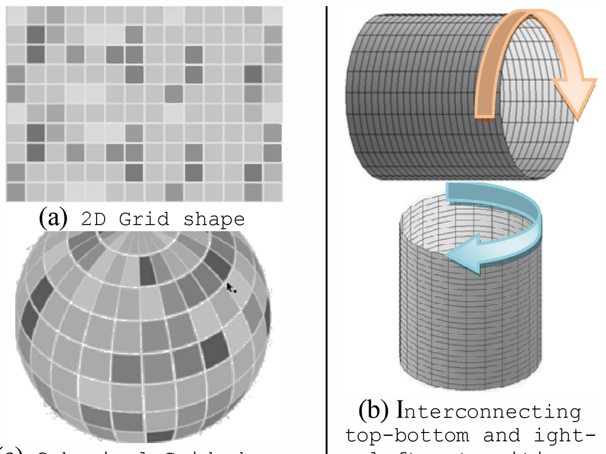

$m = \left\lfloor \frac{n}{\lfloor\sqrt{n}\rfloor} \right\rfloor$
$p = \lfloor\sqrt{n}\rfloor$
$K = n - m \times p$
$= n - \left\lfloor \frac{n}{\lfloor\sqrt{n}\rfloor} \right\rfloor \times \lfloor\sqrt{n}\rfloor$
1
2
3
4
5
6
7
8
1 a corner node
5 an internal node
4 an external node

Figure 1: Example of the 2D view of the Grid

In the second step, we connect each internal node to its four neighbors. As a result, external nodes are connected to only three neighbors and the Grid corners only to tow neighbors. Afterwards, we connect external nodes at the top (respectively, at the right) to nodes at the bottom (respectively, at the left) of the Grid. The corner nodes at the extreme right of the two last lines are connected. Connections between the Grid nodes (graphically represented in Figure 2) are resumed as follows:

- Each node (i,j) where $1 < i < p$ and $1 < i < m$ is connected to the nodes $(i-1,j)$, $(i+1,j)$, $(i,j-1)$ and $(i,j+1)$.
- If $n = m \times p$ then:
  - each node $(1,j)$ is connected to the node (p,j)
  - each node $(i,1)$ is connected to the node (i,m)
- If $n > m \times p$ then:
  - each node $(1,j)$ where $1 \leq j \leq k$ is connected to the node $(p+1,j)$.
  - each node $(1,j)$ where $k < j \leq m$ is connected to the node (p,j).
  - the node $(p+1,k)$ is connected to the node (p,m)
  - the node $(p+1,k)$ is connected to the node $(p+1,1)$
  - each node $(i,1)$ where $1 \leq i \leq p$ is connected to the node (i,m)


(a) 2D Grid shape

(b) Interconnecting top-bottom and ight-left extremities

(c) Spherical Grid shape

Figure 2: Node interconnections

The construction of the Grid structure is $\Theta(n)$ where n is the number of nodes. The total number of connections is equal to:

$$|R| = \begin{cases} 2 \times p \times m & \text{if } n = p \times m \\ n + m \times p + 1 & otherwise \end{cases}$$

In the worst case, the Grid has a total number of connections equal to $2 \times n + 1$.

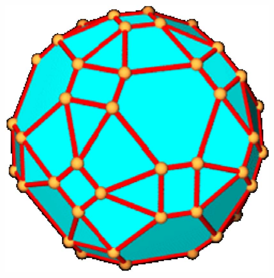


Figure 3: Polyhydric organized grid nodes

Table 1 compares the Grid topologies of ***TunGrid*** and ***Peer-to-Peer*** in terms of the total number of connections between the Grid nodes.

Number of nodes	**5**	**10**	**15**	**30**	**100**	**200**	**300**	**400**	**500**
Peer-to-Peer	10	45	105	435	4950	19900	44850	79800	124750
TunGrid	7	20	30	60	200	401	601	800	1001

Table 1: Number of node connections: Grid topology for *TunGrid* vs Peer-to-Peer.

It is very clear that ***TunGrid*** generates fewer interconnections among its Grid nodes. Evidently this influences the Grid performance.

V. LOAD BALANCING

The Grid is composed of a set of nodes. The load of the Grid is the sum of the loads of its nodes. A node's load is measured by the sum of the jobs that its queue contains; the more a node is loaded, the more its queue contains jobs. Like heat jobs spread (move) from the more-loaded nodes to the less-loaded nodes in their local neighborhood.

Two connected nodes tend to equalize their loads. By exchanging jobs they have queued up (this is called load balancing). In our new toroidal Grid topology, every node is connected logically to at least four neighboring nodes with which it shares its resources. Nodes are connected directly as peers without the need for a central routing node. Jobs keep spreading from one node to the rest of the Grid in a balanced manner until equilibrium or total absorption is reached. Because the sphere is the well adapted shape for thermal equilibrium, the spherical like topology of the Grid is well adapted for the local propagation of the overload and thus load balancing over the entire Grid. The power of the spherical shape of the Grid is thus justified.

The figure 5 shows how the Grid looks like and how the load balancing functionality works. ***TunGrid*** uses a sender side initiated algorithm. When a node becomes overloaded, a sender side algorithm is executed in order to find from its neighbors an underloaded node ready to accept one or more of its queue tasks.

In case of an unexpected peak or node failure, load balancing algorithm is executed. A job queued in the overloaded or failing node is routed to relatively idle nodes in the Grid.

Let $J_i = \{j_{i1}, j_{i2}, j_{i3}, \dots, j_{ix}\}$ represents the set of jobs in node i. If the node's queue length $|Q_i|$ exceeds a threshold α, it is considered as an overloaded node $(|Q_i| > \alpha)$. If the node's queue length $|Q_i|$ is less than or equal to a threshold β, it is considered as an under-loaded node $(|Q_i| \leq \beta)$. The thresholds α and β are chosen by the Grid administrator. Note that it is possible to choose $\alpha = \beta$

When the node i becomes overloaded, it tries to find an under-loaded node j from its group members $\Gamma(i)$ready to receive some jobs from its current queue. $\Gamma(i)$ is the set of nodes directly connected to the node i according to the Grid topology. For example in Figure 4 the overloaded node 18 can send jobs only to nodes 19, 17, 6, and 15 whereas the node 7 can send to 19, 6, 3, 26. In the illustrated case the nodes 19 and 6 are common neighbors of the nodes 7 and 18. In case of a conflict; when the nodes 18 and 7 try at the same time to send jobs to one of their common neighbors 19 or 6, a FIFO strategy is used to resolve the conflict.

At any instant, the Grid nodes have different levels of load and tend to balance their loads with their neighbors. If a node i and all its neighbors $\Gamma(i)$ are overloaded, then i (colored in red in Figure 4) needs to wait until one of its neighbors (colored in green in Figure 4) becomes underloaded.

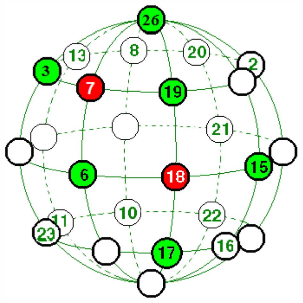


Figure 4: Example of Grid load situation

A situation of famine occurs when one part of the Grid is totally overloaded while the rest is totally underloaded or not being used. Every time a new job arrives or due to an execution failure, a job needs to be rescheduled: ***TunGrid*** selects randomly an available node to execute it. This random assignment reduces the risk to get into a situation of famine where the resource load is concentrated in a small number of the nodes where the others are not being used.

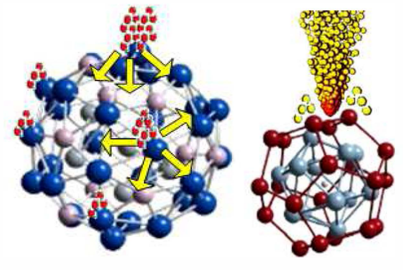


Figure 5: Jobs spreading from over-loaded nodes to under-loaded nodes in a balanced manner

VI. EXPERIMENTATION AND RESULTS

In order to validate ***TunGrid***, we used the *GSSIM* simulator [2] (Grid Scheduling Simulator). It is a simulation framework based on GridSim, which provides an easy-to-use Grid scheduling framework for enabling simulations of a wide range of scheduling algorithms with diverse granularity and scope in multi-level heterogeneous Grid infrastructures. GSSIM is structured with a set of flexible and replaceable plugin components, such as *Grid scheduling plugins*, *local scheduling plugins*, and *runtime calculation plugins*.

We had to develop and modify several internal structures and interfaces to fulfill specific requirements of *TunGrid* and take into account semantics for nodes description and different resource types. We grouped the developed components in the ***TunGrid package***.

We used GSSIM simulator with different scheduling algorithms: *TunGrid, Round robin,* and *Shortest First*. We created varied simulations with 100, 1000, and 10000 tasks and 5, 10, 15, and 20 resources.

a) Failed requests

As illustrated in Figure 6, the simulation results show that the ***ShortestFirst*** algorithm causes a very important number of failed requests during all different simulations. The failures are due to an incompatibility between job requirements and resource characteristics.

This type of algorithms is called blind scheduling. Every time a new task is received, the ***ShortestFirst*** scheduler allocates it to a random resource without checking its requirements. If the task execution fails, the task is rescheduled.

In contrast, the ***TunGrid*** and RoundRobin scheduler cycle start by making a pre-resource selection to eliminate all incompatible resources.

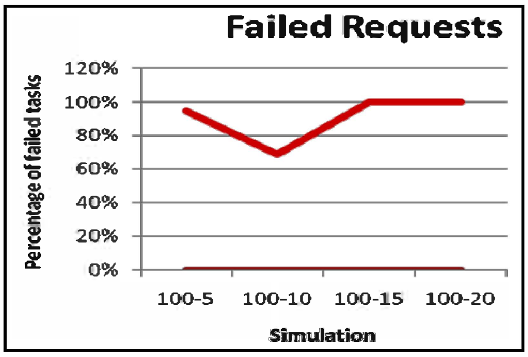


Figure 6: Failed requests

b) Total resourceload

The second evaluation criterion is the total resource load (TRL). Figure 7 and Figure 8 represent the minimum, maximum, and mean variance values of the TRL respectively.

The simulation results show that: first, the *Round Robin* algorithm has the maximum and minimum utilization values. This means that some of the resources are overloaded while some others are not used. Second, the *ShortestFirst* algorithm shows a total failure when used with more than 5 resources and 100 tasks. Third, ***Tungrid*** shows the best results in terms of resource load distribution. All resources are being used. There is neither overload nor starvation.

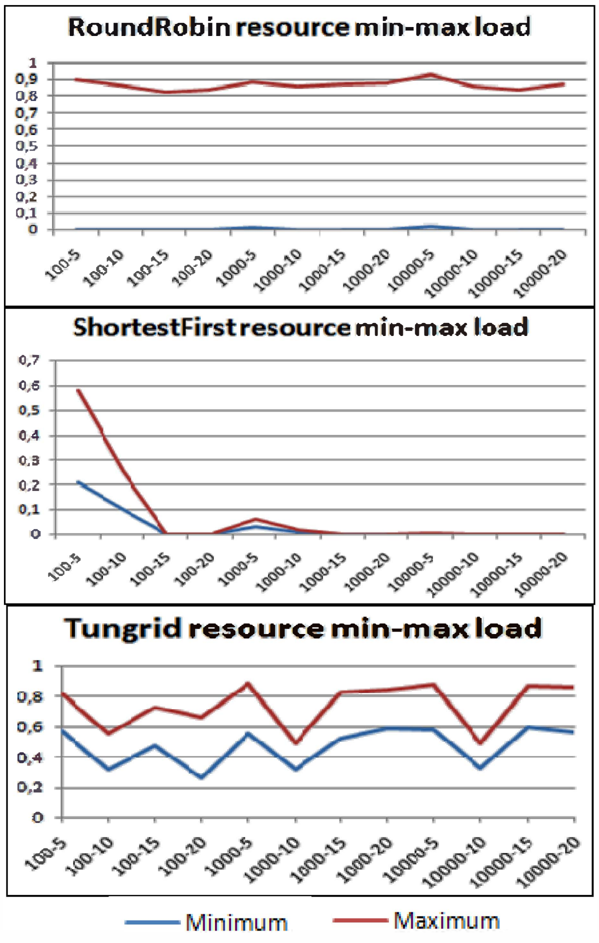


Figure 7: Minimum and maximum resource load

The simulation results show that:

1. The *RoundRobin* algorithm has the maximum and minimum utilization values. This mean that some of the resources were overloaded while others were not used
2. The *ShortestFirst* algorithm shows a total failure when used with more than 5 resources and 100 tasks.
3. *TunGrid* shows the best results in term of resources were being used. There is neither overload nor starvation.

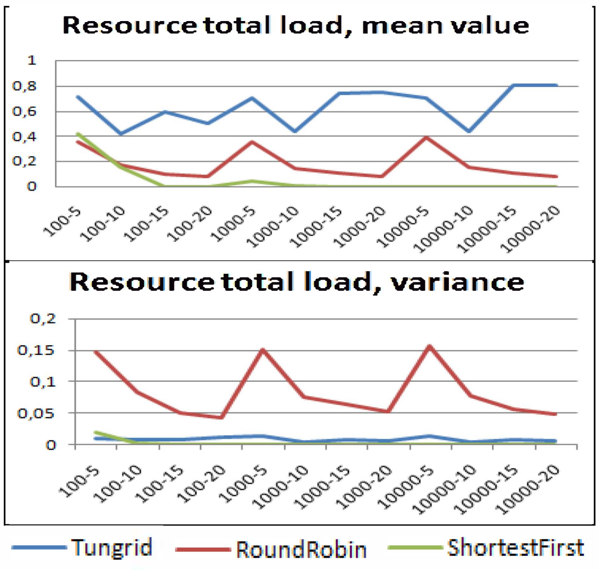


Figure 8: Resource load mean value and variance

The simulation results shown by the figures 7 and 8 demonstrate clearly that TunGrid gives on average the best resource load (between 40% and 80%) while RoundRobin and ShortestFirst utilization rate is under 40% during all simulations. In addition, TunGrid resource load variance is better than RoundRobin's; it is lower than 0.02.

c) Speed up

Figure 9 shows the evaluation of the speed up while increasing the number of resources. The speed up realized by ***TunGrid*** (up to 22%) is always higher than RoundRobin (between -1% and 4%).

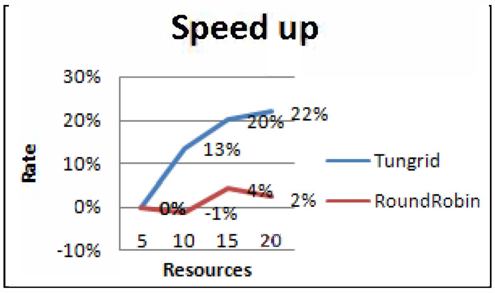


Figure 9: Algorithm speed up.

d) Energy Usage

The Figure 10 shows the mean value of energy usage. It is very clear that, compared to Round Robin and Shortest First, ***TunGrid*** consumes more important quantities of energy. The *SortestFirst* algorithm consumes the minimum energy.

Figure 11, which illustrates the minimum and maximum values of energy usage that some resources are not consuming energy when we use RoundRobin. This is due to their lack of utilization.

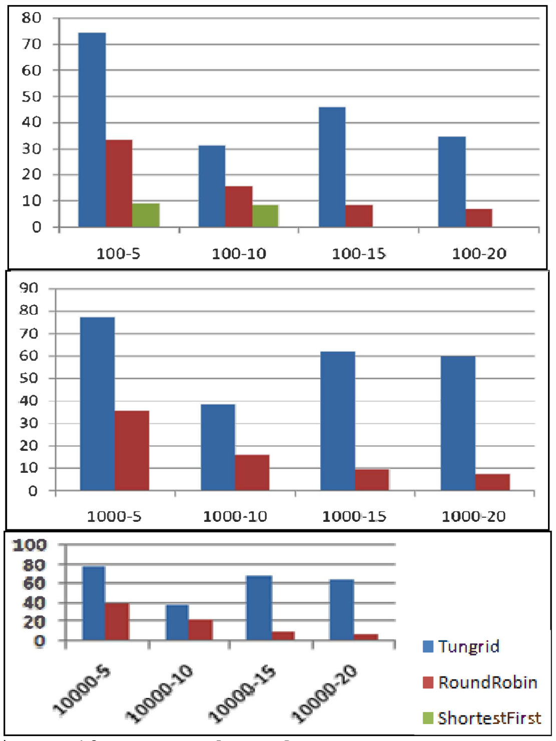


Figure 10: Mean value of energy usage

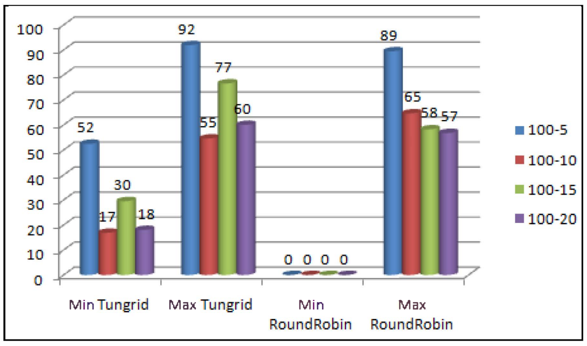


Figure 11: Minimum-Maximum values of energy usage

Finally, one might emphasize from these simulation results ***TunGrid*** has shown good results in terms of resources load, and speed up. Even though it meets the development goals, one might also notice that the solution just found is not really the optimal that TunGrid can always be improved.

VII. Conclusion

This work dealt with the development of an innovative fully decentralized Grid meta-scheduling method for non dedicated resources. The proposed algorithm which we called ***TunGrid*** mimics the heat propagation phenomenon: the Grid is modeled through a toroidal shape (that can be constructed in $\Theta(n)$for a Grid with $n$nodes) which is a well-adapted shape for an optimal heat propagation until a balanced coverage.

Based on this natural phenomenon, load balancing is conducted through task transfer between immediately connected nodes. We showed that our toroidal construction algorithm produces much fewer connections than peer-to-peer. Furthermore, through this meta-scheduling method, we illustrated how the Grid scheduling techniques are useful to improve the Grid performance by offering better resource load balancing, execution time, failed requests rate, speed up, and energy consuming.

To demonstrate the advantages of ***TunGrid***, we used the GSSIM Grid simulator to create simulations using different number of resources and tasks.

The simulations showed encouraging results in terms, of both resource loads, and execution speed up. It meets the following Grid scheduling algorithm goals full decentralization, reliability, efficiency, seamless integration, parallel job execution, resource balancing, infinite scalability, and inter-organizations collaboration.

We are working on optimizing ***TunGrid*** to support more parameters for instance to provide for the specification of dedicated resources or special hardware, *e.g.*, supercomputers, high storage capacity devices, storage depot, and special multimedia devices and software to the Grid. In addition we are examining how ***TunGrid*** can support dynamic reconstruction of the Grid in case of resource unavailability or addition.

References

[1] Danilo Ardagna, “Autonomic services: An introduction Grid-Scheduling”, Department of electronics and informatics, Polytechnic of Milano, Italy, November 2007

[2] Andrea Pugliese, Domenico Talia and Ramin Yahyapour, “Modeling and Supporting Grid Scheduling”, Journal of Grid computing, July 2007

[3] Fangpeng Dong and Selim G. Akl, “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”, School of Computing, Queen’s University Kingston, Ontario, Canada, January 2006

[4] Apostolos Gerasoulis and Tao Yang , “A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors”, Journal of Parallel and Distributed Computing, Volume 16,Issue 4, pp.276-291, Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903, USA, December 1992

[5] Stefka Fidanova, Mariya Durchova, “Ant Algorithm for Grid Scheduling Problem”, Large Scale Computing, Lecture Notes in Computer Science No 3743, Bulgaria, 2006

[6] Marcin Krystek, Krzysztof Kurowski, Ariel Oleksiak, and Krzysztof Rzadca, “COMPARISON OF CENTRALIZED AND DECENTRALIZED SCHEDULING ALGORITHMS USING GSSIM SIMULATION ENVIRONMENT”, Poland, Poznan Supercomputing and Networking Center and Grenoble University, Spring 2008

[7] Fangpeng Dong and Selim G. Akl, “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”, School of Computing, Queen’s University Kingston, Ontario, Canada, January 2006

[8] Satu Elisa Schaeffer, “Graph clustering”, Computer Science Review, Laboratory for Theoretical Computer Science, Helsinki University of Technology TKK, Finland, August 2007

[9] Jens Hildebrandt, Frank Golatowski, Dirk Timmermann, “Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems”, Institute of Applied Microelectronics and Computer Science, Department of Electrical Engineering and Information Technology, University of Rostock, Germany, June 1999

[10] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen & Richard F. Freund, “Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems”, The 8th Heterogeneous Computing Workshop (HCW ’99), IEEE Computer Society, San Juan-Puerto Rico, April 1999

[11] Antonella Galizia, Alfonso Quarati, "Job allocation strategies for energy-aware and efficient Grid infrastructures", Journal of Systems and Software Volume 85, Issue 7, July 2012, Pages 1588–1606

[12] Wei-Chang Yeh, Shang-Chia Wei, "Economic-based resource allocation for reliable Grid-computing service based on Grid Bank",Future Generation Computer Systems Volume 28, Issue 7, July 2012, Pages 989–1002

[13] Haruna Ahmed Abba, Syed Nasir Mehmood Shah, Nordin B. Zakaria, Anindya.J.Pal, "Deadline Based Performance Evaluation of Job Scheduling Algorithms", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discover, Sanya, China, 2012

[14] Rajkumar Buyya, Manzur Murshed, David Abramson, “A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids”, Technical Report, Monash University, March 2002

[15] S Saeid Abrishami, Mahmoud Naghibzadeh, Dick Epema, “Cost-driven scheduling of grid workflows using partial critical paths”, in 11th IEEE/ACM International Conference on Grid Computing, 2010.

[16] “THESEUS-FE 4.2 Validations Manua“, P+Z Engineering GmbH, Munich ,Grmany, june 2012.

[17] Zoltan Spakovszky, “Heat Transfer”, Thermal-Fluid Systems course, Unit 5: Heat Exchangers, MIT