# Dynamic on Demand Virtual Clusters in Grid

Mario Leandro Bertogna[1], Eduardo Grosclaude[1], Marcelo Naiouf[2], Armando De Giusti[2], and Emilio Luque[3]

[1] Department of Computer Science
Universidad Nacional del Comahue
C.P. 8300. Buenos Aires 1400. Argentina
{mlbertog,oso}@uncoma.edu.ar
[2] Informatic Research Institute LIDI
Universidad Nacional de La Plata, Argentina
{mnaiouf,degiusti}@lidi.info.unlp.edu.ar
[3] Computer Architecture and Operating System Department
Universidad Autónoma de Barcelona, Spain
Emilio.Luque@uab.es

**Abstract.** In Grid environments, many different resources are intended to work in a coordinated manner, each resource having its own features and complexity. As the number of resources grows, simplifying automation and management is among the most important issues to address. This paper's contribution lies on the extension and implementation of a grid metascheduler that dynamically discovers, creates and manages on-demand virtual clusters. The first module selects the clusters using graph heuristics. The algorithm then tries to find a solution by searching a set of clusters, mapped to the graph, that achieve the best performance for a given task. The second module, one per-grid node, monitors and manages physical and virtual machines. When a new task arrives, these modules modify virtual machine's configuration or use live migration to dynamically adapt resource distribution at the clusters, obtaining maximum utilization. Metascheduler components and local administrator modules work together to make decisions at run time to balance and optimize system throughput. This implementation results in performance improvement of 20% on the total computing time, with machines and clusters processing 100% of their working time. These results allow us to conclude that this solution is feasible to be implemented on Grid environments, where automation and self-management are key to attain effective resource usage.

## 1 Introduction

Using geographically distributed clusters in a coordinated manner has a major impact in execution time for parallel applications. Grid computing is a natural environment to deal with this usage, as Grids provide resource sharing through open standards and tight security, making possible to solve problems faster and efficiently.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 13–22, 2009.

The Grid metascheduler is an active component in distributed systems coordination and management. This component facilitates the user's tasks to access resources across different administrative domains. It can take decisions based on information of the whole system. Owners of physical resource become services providers, and the metascheduler orchestrates them according to negotiated policies and service level agreements. Virtual machines provide a way to make this task easier. These virtual machines can be independently instantiated and configured beforehand with sandbox-like environments[4, 9]. They also allow dynamically tuning of parameters like memory, number of CPUs assigned to each virtual machine, etc. Today's virtual machine technologies performance at CPU intensive tasks is comparable to that of native applications [5].

This paper presents a framework extension to a Grid metascheduler. The extension consist of two modules; the first one that dynamically discovers free machines determined by user requirements. The other one creates virtual clusters to efficiently satisfy submition of parallel jobs. The first module selects the clusters using graph heuristics. Free resources are mapped to a graph, machines as nodes and network links as edges. The algorithm then tries to find a solution by searching a set of clusters that achieve the best performance for a given task. The second module, of which one instance resides in every grid node, monitors and manages physical machines. When a new task arrives, these modules modify virtual machines configuration to dynamically adapt resource distribution at the clusters, thus obtaining maximum utilization. Metascheduler components and local administrator modules work together to make decisions at run time to balance and optimize system throughput.

In the second section of this paper we present the sequence of use and the architecture of the solution, focusing on the metascheduler; the third section describes the model, heuristics and algorithms developed; the fourth section presents experimental results; and finally, related works on this subject and conclusions are shown.

## 2 Metascheduler

From the design viewpoint, the architecture[1] of this solution is conceptually divided into three layers or tiers. In the first one, named *access tier*, the clients accessing the system are defined. The second, *management tier*, considers access control and creation of resources. Finally, the third, *resource tier*, deals with the implementation of physical and virtual resources.

This paper focus on the management tier cover by the metascheduler. The implementation begins with the study of several proposals, for this work, CSF (Community Scheduler Framework)[2] was chosen. CSF is an open-source implementation of a number of Grid services, which together functionally perform as a Grid metascheduler, and can be used as a development toolkit.

To satisfy on demand virtual clusters there are two extension modules within CSF. The first one is the *Resource Manager Adapter Service* called *GramVM*. This module is in charge of looking for free machines in the group of clusters.

For management and instantiation of virtual machines within local domains, a new module called *Hypervisor proxy* was implemented. Unlike the original CSF proposal, several *Hypervisor proxy* instances can be working towards one *GramVM* instance at the same time, in a coordinated manner. In the original implementation, CSF could work with different local schedulers, just one at every time.

Also different from the original CSF is that local schedulers, with similar duties as *Hypervisor Proxy,* have to be setup previously to CSF execution, and once execution in a cluster is started, the assignment of machines can not be modified until end of execution. For dynamically instantiated virtual machines, we do not know how many machines each cluster will have until the requirement arrives. *GramVM* should try to find the resources that best fit the task requirements, leading to a great number of alternatives. Besides, virtual machines can modify their usage of physical resources during execution, either by live migration[6], or by dynamically varying memory and CPU allocation. All of these features essentially reconfigure the pool of available free resources. They can be used to obtain better cluster performance and they are negotiated between *Hypervisor Proxy* and *GramVM* at run time.

## 3 Model

Finding a group of machines with specific characteristics, which is able to efficiently share a given workload, in a short time, is not a trivial problem. To approach this task, we settled for two criteria which were given higher priority: how fast the problem was solved, and how good the outcome was when compared to the optimal solution.

To be able to solve the problem in an analytical way, groups of clusters and free machines in the Grid environment are mapped to a graph. Machines are viewed as nodes and network links as edges. Nodes and edges have weights corresponding to machine features and bandwidth. More bandwidth-capable edges receive less weight. Node's weights are based upon cost functions such as per-time billing, computing power, etc.

The strategy is divided into two stages. The first one consists in selecting the groups of clusters. At this stage, a heuristic is used to find an optimal set of machines, taking into account communication overhead and machines computing power. Once a group of clusters is obtained, the second stage starts. For each cluster, an analysis must be done to evaluate how many physical machines will be incorporated. If the number of machines involved is greater than needed then efficiency will decrease, as some machines will stall waiting to send data to another cluster. We seek to keep efficiency over a certain threshold, given beforehand. Our analysis extends the work done in [3]. This work modifies the MPI library to span a number of clusters; a certain, unique, type of task is assumed. In our paper, a virtual environment is proposed where applications can run unmodified over a combination of clusters. Different types of tasks can be supported, as expected from a Grid environment.

Our model was evaluated over master-worker parallel applications. Clusters are dedicated and serve a previously determined number of tasks. All tasks within a same requirement from a user perform the same computation, and send or receive the same amount of data, but the number of tasks can vary across requirements. This schema is common in graphic and simulation parallel applications. It is assumed that each time a cluster is added to the grid environment, virtual images are characterized and a performance benchmark is done. The network links are monitored regularly to sense bandwidth changes.

### 3.1 Machine Selection Algorithm

To select a set of feasible clusters to be incorporated into the solution, an iterative improvement algorithm is used. This algorithm, known as *Hill-Climbing*, is mainly a loop that continually moves in the direction of increasing value; in this case, the direction maximizing computing power. The algorithm does not maintain a search tree; the node data structure needs to record just the last state reached. This simple policy has some drawbacks: *local maxima* (peak values that are lower than the highest peak value in the state space); *plateaux* (a region in the state space where the evaluation function is essentially flat) and *ridges* (a ridge may have steeply sloping sides, so that the search reaches the top of the ridge with ease, but the top may slope only very gently toward a peak).

The problem we are trying to solve has particular features, as certain networked geographical regions or provider domains are better provisioned than others. This geographical connectivity pattern is mapped onto the graph edges. When taking this feature into account, there is no need to do random restarts as in the original algorithm. If the graph is partitioned into better-connected geographical zones, or islands, the chances to find the global maximum grow, and the time to find it decreases. This modification is called *Hill-Climbing with k-restarts*, where $k$ is the number of partitions on the graph. Each partition will be a starting point.

To partition the graph in geographical zones, a different kind of algorithm is used, namely *Minimum Spanning Trees (MST)*. A minimum spanning tree includes all nodes in the graph, such that the sum of their weighted edges is lesser or equal to that of any other spanning tree over the graph. The chosen algorithm is Kruskal's variant because of the approach taken to build the MST. This algorithm starts by sorting the edges by weight; then all nodes are agglomerated, starting from as many partitions as nodes. Traversing over the edges, the solution is checked at every iteration for cycles. If a cycle appears, the edge that was most recently introduced is discarded.

To enhance *Hill-Climbing* performance, we need to know how many restarts there will be. The number of restarts will be the number of suitable partitions in the graph. Once the number of partitions is set as a threshold, *Kruskal*'s algorithm starts adding edges until the threshold is reached. When *Kruskal* algorithm stops, the remaining graph is partitioned into maximally well-connected trees, as the first step taken was to sort the edges by weight. For each partition the *Hill-Climbing* algorithm is then applied, obtaining a global maximum.

The number of partitions depends on the number of nodes and the surface where *Hill-Climbing* algorithm will run. As a good practice the graph was partitioned until each segment had at least one complete cluster. In most cases that number of partition was three, this number assures to find the global maximum in each test.

The algorithmic complexity for MST is *O(E log(V))*. The *Hill-Climbing* algorithm using adjacency lists is *O(E log(V))* where *E* are Edges and V are Vertices of the graph. Performance can be improved if the graph nodes are Grid nodes instead of machines, as the number of vertices in the graph decreases.

### 3.2 Cluster Usage Optimization

A parallel application in a multicluster environment is either limited by performance of machines in the cluster (compute-bound) or by network throughput (communication bound). The maximum performance (*maxperf*) is reached by an application on a particular cluster when it is compute-bound. If the application is communication bound, machines will sit idle waiting for network input/output. For a worker task running on a processor, the computation time ($T_{Cpt}$) is defined as the ratio between the task number of operations (*Oper*) and the processor performance (*Perf*): $T_{Cpt}=Oper/Perf$. The communication time ($T_{Comm}$) is the ratio between the volume of data communication (*Comm*) (worker task data from and to the master) and the network throughput ($T_{Put}$): $T_{Comm}=N*Comm/T_{Put}$. The *maxperf* is the performance that can be obtained when $T_{Cpt} >= T_{Comm}$.

Once the set of clusters is computed by the heuristics, the second stage starts. Here we analytically determine how many machines will be used in each cluster, so as to avoid *maxperf* dropping under a previously fixed threshold. This calculation is based on how many task's data the network is able to transfer by time unit, and how many tasks per time unit the cluster is able to process.

If the cluster processes more tasks than the network would transfer, then the application becomes communication bound; if the network is able to transfer more task's data than the cluster processes, then the application becomes computation bound. Hence, if the application is communication bound and the number of machines diminishes until a balance is reached, the cluster resources are not fully used; but the machines processing the tasks will be used at a maximal efficiency.

In a Grid environment, multiple possibilities for cluster assignment exist. If we regard the application as being started from different clusters (i.e. we choose different *master-clusters*), the resulting outcome from our analysis will be different. With the heuristic algorithm, this search work is minimized, and the best solution (reaching the highest computing power with a combination of clusters) is probably achieved. To make this possible, an analytic search has to be done for each graph partition. This will limit the number of machines for each cluster so that performance can be held over threshold for every machine. Not only communication and computing power for the local cluster has to be evaluated, but for the *master-cluster* as well. If the *master-cluster*'s bandwidth is smaller than the aggregated *worker-clusters* bandwidth, then machines from the

*worker-clusters* will be idle even though their own network links are enough to exploit their full computing capacity for a given task. Here, a fractional of the *master-cluster* bandwidth will be determined, based on the computing power of each cluster; and the analytic evaluation of computation/communication will be carried on upon this value.

This solution focuses on maintaining machine performance, but cluster usage is also a matter of importance. If a cluster can always satisfy a task with low computing requirements but high data communication, and the network link does always limit the computing power to a few machines, then this is not a good solution. An approach to this problem is to limit even more the usage of computing power, so as to free bandwidth. When tasks with less communication requirements arrive, they can be submitted to idle machines, thus improving cluster usage; on the downside, if such tasks never arrive, resources will be wasted. Our proposal in this paper is to build a cluster over virtual machines to release bandwidth on demand; when a new task with smaller communication requirements arrives, machines already executing on the clusters are migrated without interrupting the execution. When two or more virtual machines are executing on a physical processor, the virtualization software scheduler will assign computing resources fairly, so this will result in less computing power per machine and less data per task will be sent. The time for completion of both the new task and the executing task are known, so the metascheduler module can calculate the time gain for machine migration and will submit the new task onto the cluster. If the new task has smaller communication requirements than the migrated one, not only the physical nodes that were supporting virtual machines running on them, but also the idle physical nodes in the cluster, could be assigned to the new task, improving the whole cluster usage.

## 4 Experimental Results

The experimental evaluation is divided into two phases. The first one shows improvement gains by using the heuristic of machine selection. This strategy is compared to classical grid algorithms[7, 8] as a set of independent tasks arrives. From the system's point view, a common strategy is to assign them according to the load of resources in order to achieve high system throughput. Three algorithms were selected: a) Minimum Execution Time (MET): assigns each task to the resource with the best expected execution time for that task, no matter whether this resource is available or not at the present time b) Minimum Completion Time (MCT): assigns each task, in an arbitrary order, to the resource with the minimum expected completion time for that task and c) Opportunistic Load Balancing (OLB): assigns each task, in arbitrary order, to the next machine that is expected to be available, regardless of the task's expected execution time on that machine . The intuition behind OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity, but because OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans. Classical grid algorithm like Min-min and Max-Min were not

selected because these begins with the set of all tasks and in this case this data is unknown.

The problem of grid scheduling can be investigated by taking experimental or simulation approach. The advantages of performing actual experiment by scheduling real applications on real resources are that it is easier and straightforward to compare the efficacy of multiple algorithms. However, in experimental study of scheduling algorithms it is seldom feasible to perform a sufficient number of experiments to obtain meaningful results. Furthermore, it is difficult to explore a variety of resource configurations. Typically a grid environment is highly dynamic; variations in resource availability make it difficult to obtains repeatable results. As a result of all these difficulties with the experimental approach, simulation is the most viable approach to effectively investigate grid scheduling algorithms. The simulation approach is configurable, repeatable, and generally fast, and is the approach we take in this work. Our simulation takes as parameters a description of the existing clusters, their network links, machines therein (specifying memory and processor type) and finally the tasks with their execution time for each type of processors.

For model validation a real test in [3] was considered, where four clusters with three, five and eight machines were used. The first two clusters were physically lying in South America, having less bandwidth and machines with smaller computing power. The third, more powerful one, was in Spain. If we enforce the same *master-cluster* as in the real test, i.e. the application is submitted from each of the clusters in South America, the final results in computing time and network throughput returned by the simulator are the same. However, if the simulator is used along with the machine selection algorithm, the Spanish cluster is selected and the total execution time decreases nearly by 50%. The explanation being, if the application is submitted from a South American cluster, then there will be idle machines in the Spanish cluster; while if the application is submitted from Spain, then the three clusters will have better usage.

### 4.1 Machine Selection Experiences

Tests were done to verify the impact of partition and *master-cluster* selection carried over by the *Hill-Climbing* algorithm. These tests compare how the heuristic algorithm proposed in this paper performs against classical grid algorithms. The tests were done simulating clusters composed of eight machines each one. Two arrival statistical distribution were chosen; the first one, a uniform distribution simulating low rate of arrivals; the other one, an exponential distribution simulating a incremental rate of tasks arrivals. Nineteen request were made, each one with three hundred process to be distributed into the clusters. In all cases MCT performs better than MET and OLB. For greater clarity, figure comparisons were done between Hill-Climbing and MCT algorithms.

The results of simulation executions can be seen in figure 1. In the firts graph, *Hill-Climbing* uses all clusters machines, selecting the best master cluster and MCT only selects the minimum completion time cluster. Choosing to run the application in all machines heuristic algorithm performs 70% better than MCT,

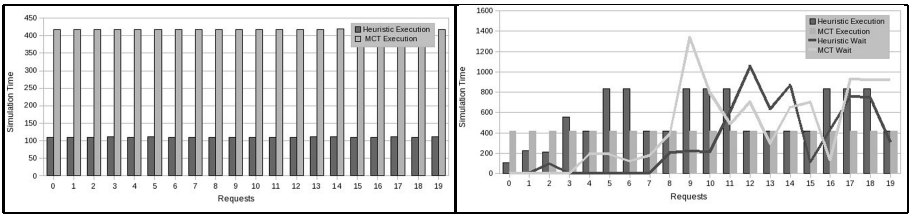


**Fig. 1.** Time execution comparison between Heuristic and MCT algorithm with uniform task arrival and exponential task arrival

this one is the best case. If each parallel task sends more data while processing, then the application becomes communication bounded ($T_{Cpt} <= T_{Comm}$) limiting the number of machines each cluster could use to compute, decreasing the algorithm performance. Once the algorithm proved to work with low rate of arrival the next step was to test it incrementing this rate. This can be seen in the second graph. Until the second request the execution time is below MCT. For some request the execution time are above MCT. This happens because tasks are distributed over different clusters. If tasks waiting times are observed (shown as continuous lines), MCT has longer waiting periods, compensating for faster executions. The average total time tends to be the same. After several simulations we can conclude that in the worst case *Hill-Climbing* heuristic tends to perform the same as MCT. In the average scenario (low rates of arrival with peaks at regular intervals), the *Hill-Climbing* heuristic algorithm performs with a 20% of improvement.

### 4.2 Cluster Adaptation Experiences

In cluster adaptation test, different task types were submitted requiring differents volumes of I/O. Task mixes were done with exponential rate of arrivals. Some requests overlap in time but waiting times were not too high. Because of this they are not shown in the graphs.

After several simulation executions it was noticed that some clusters can not be 100% used because tasks with high network I/O requirement took little percentage of clusters machines while saturating Internet connection. In the case where tasks with low network requirements arrive, this can be a waste of computing resources (as blocked machines could be used with smaller bandwidth). To solve this problem a migration procedure was proposed, if a virtual machine consuming network bandwidth was migrated to other physical machine already working, taking advantage of virtual machine live migration. The fair scheduling algorithm used in hypervisors will slow down computing power freeing network bandwidth, and these free bandwidth will be taken by the new process. This procedure is advantageous if the time waste in the migrated task is less than the time gain processing the new task.

In figure 2 we depict *Hill-Climbing* heuristic execution time and MCT execution time with a similar behavior as the previous figure with a 25% of improvement, and

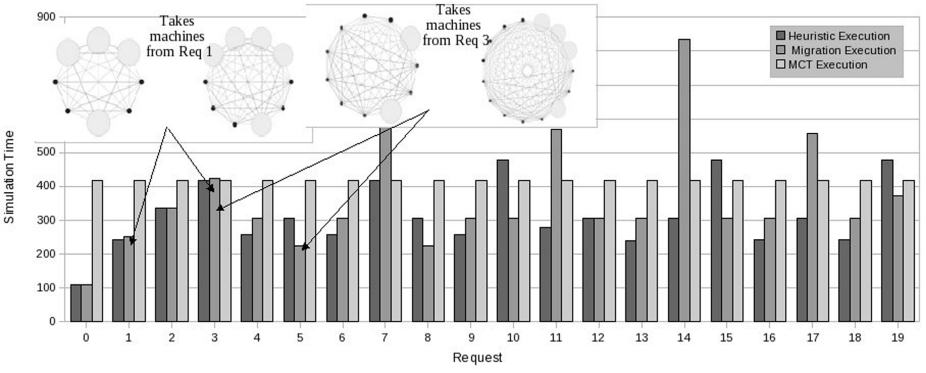


**Fig. 2.** Migration process sample

an example of the migration procedure. When Req3 arrives, the algorithm investigates a) where machines can be migrated and b) what the impact should be on the running application, were these machines actually migrated, i.e. whether the gain upon migration would pay for migration overhead. These conclusions can be drawn only having the application's run time characterized beforehand. In the boxes of figure 2 two examples of this procedure can be seen. In the upper left part of the box, a graph is shown with the machines originally assigned to the task (in this case Req3 and Req5) and then the machines after the virtual machine migration procedure. In the first case eight machines were assigned and then two machines from blocked clusters were added. A small increment can be noticed in the time in the execution of Req1 because of the slow down in the computing power after the migration. The same can be seen in Req5. The time gain is more obvious there. In the rest of the simulation, several cases exist with a longer execution time. This ocurrs because of different configuration of machine assignment after the migration process. On average, the performance improvement is of 5% over the originally proposed algorithm but this depends on tasks balance and arrival rate.

## 5 Conclusion

This paper has presented a metascheduler framework extension to generate high performance laboratories with virtual machines, local resource managers and management heuristic to obtain effective usage of clusters and machines. The framework takes into account not only the time taken by a task to complete but also network consumption, with the purpose of taking advantage of the bigger number of machines available in a grid environment. Geographical partitions through *Kruskal* graph algorithm also address the problem of scalability, decreasing the complexities in the search of the optimal solution. *Hill-climbing* with *k-restarts* does not ensure reaching an optimal solution; but in the tests done the best solution was achieved in nearly every case.

Tests were done by sweeping a range of arrival rates, cluster computing power, number of tasks, number of process per tasks and task computing and I/O requirements. Selection algorithms have been implemented to find groups of clusters that satisfy certain requirements in a small search space, making an effort to return a solution in a fast and optimal way. These implementations increase computing power by nearly 20%. Dynamic algorithms have been implemented to adapt cluster configuration at run time with migration of virtual machines. This implementation results in performance improvement of 10% on the total computing time, with machines processing 100% of their working time.

These results allow us to conclude that this solution is feasible to be implemented on Grid environments, where automation and self-management are key to attain effective resource usage. Where clusters serve fixed applications, multicluster analysis could guide balance tuning between computation and communication, determining whether it is more effective to either increment/decrement the bandwidth in use, or increment/decrement engaged computing power.

## References

[1] Grosclaude, E., Luro, F.L., Bertogna, M.L.: Grid Virtual Laboratory Architecture. In: VHPC Euro-Par 2007 (2007)

[2] Open source metascheduling for Virtual Organizations with the Community Scheduler Framework (2004), `http://www.cs.virginia.edu/~grimshaw/CS851-2004/Platform/CSF_architecture.pdf`

[3] Argollo, E., Gaudiani, A., Rexachs, D., Luque, E.: Tuning Application in a Multicluster Environment. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 78–88. Springer, Heidelberg (2006)

[4] Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual Clusters for Grid Communities. In: CCGrid 2006 (2006)

[5] Barham, P.T., Dragovic, B., Fraser, K., Hand, S., Harris, T.L., Neugebauer, R.: Xen and the Art of Virtualization. In: SOSP 2003, pp. 164–177 (2003)

[6] Clark, C., Fraser, K., Hand, S., Hansen, J.G.: Live Migration of Virtual Machines. In: Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (2005)

[7] Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems,Technical Report No. 2006-504, Queen's University, Canada, 55 pages (2006)

[8] Zhu, Y.: A survey on grid scheduling systems, Technical Report, Computer Science Department of Hong Kong University of Science and Technology (2003)

[9] Ruth, P., McGachey, P., Xu, D.: VioCluster: Virtualization for Dynamic Computational Domains. In: Proceedings of the IEEE International Conference on Cluster Computing, Cluster 2005 (2005)