Transparent Resource Allocation to Exploit Idle Cluster Nodes in Computational Grids

Marco A. S. Netto[1], Rodrigo N. Calheiros[2], Rafael K. S. Silva[2]
César A. F. De Rose[3], Caio Northfleet[4], Walfredo Cirne[5]

[1]Dept. of Computer Science, University of São Paulo - São Paulo, Brazil
[2]Research Center in High Performance Computing - Porto Alegre, Brazil
[3]Catholic University of Rio Grande do Sul - Porto Alegre, Brazil
[4]Hewlett-Packard Brazil - Porto Alegre, Brazil
[5]Federal University of Campina Grande - Campina Grande, Brazil

netto@ime.usp.br,{rodnc, rafael}@cpad.pucrs.br
derose@inf.pucrs.br, caio.northfleet@hp.com,walfredo@dsc.ufcg.edu.br

Abstract

Clusters of workstations are one of the most suitable resources to assist e-scientists in the execution of large-scale experiments that demand processing power. The utilization rate of these machines is usually far from 100%, and hence this should motivate administrators to share their clusters to Grid communities. However, exploiting these resources in Computational Grids is challenging and brings several problems. This paper presents a transparent resource allocation strategy to harness idle cluster resources aimed at executing grid applications. This novel approach does not make use of a formal allocation request to cluster resource managers. Moreover, it does not interfere with local cluster users, being non-intrusive, and hence motivating cluster administrators to publish their resources to grid communities. We present experimental results regarding the effects of the proposed strategy on the attendance time of both cluster and grid requests and we also analyze its effectiveness in clusters with different utilization rates.

1. Introduction

There are several techniques in which researchers can obtain third-party computational resources to their high performance applications. One of these techniques is opportunistic computing, e.g., exploiting resources even if they are partly available [20]. As examples there are public computing [1] projects, such as SETI@home [2], which work with desktop machines donated by volunteers. Other similar projects, such as Folding@home [19], FightAIDS@home [10] and Distributed.net [8], use the same method to obtain high performance computing power.

Grid computing, otherwise, makes possible the utilization of computational resources, such as desktop machines, clusters, supercomputers and scientific instruments, spread over the world [13]. Computational Grids can also be viewed as a bunch of distributed resources spread over several sites able to execute scientific applications. Thus, these applications are not only tightly-coupled applications, but they are also sets of independent experiments, such as Parameter Sweep Applications, aimed at performing simulations on several science fields.

Clusters of workstations are one of the most suitable resources to assist e-scientists in the execution of large-scale experiments that demand processing power. The utilization rate of these machines is usually far from 100%, especially in big machines located in universities or research centers. Even when cluster utilization is high, there are idle resources that may not be allocated to the local demand due to some workload characteristics or due to scheduler limitations – a problem known as external fragmentation in resource management. However, exploiting these resources in Computational Grids is challenging and brings several problems. For example, in order to access clusters in grids, users should have access rights to them, and in some cases, they should inform the amount of time and nodes necessary to execute their applications. This additional information is needed because, unlike desktop machines, clusters and supercomputers are in general space-shared among sev-

Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05)
0-7695-2448-6/05 $20.00 © 2005 IEEE

eral users. Moreover, allocation time of local cluster users would be reduced if cluster administrators published the resources to grid communities.

In this paper we describe a transparent resource allocation strategy to harness idle cluster resources in Computational Grids. The proposed strategy is based on opportunistic computing techniques and does not make use of a formal allocation request to cluster resource managers. Furthermore, it does not interfere with local cluster users, being non-intrusive, and hence motivating cluster administrators to publish their resources to grid communities. At the current stage of this work we designed the strategy to deal with applications composed of tasks that would not be interrupted due to a loss of resources. Thus, application models such as Parameter Sweep, Sequential or even Master-Worker Applications are suitable for the proposed strategy[1]. In this paper we also present experimental results regarding the effects of the proposed strategy on the attendance time of cluster and grid requests and we also analyze its effectiveness in clusters with different utilization rates.

The remainder of this paper is organized as follows: Section 2 presents some challenges on using clusters in Computational Grids and the motivation for the proposed allocation strategy taking into account the available solutions; Section 3 describes the transparent allocation strategy in detail, as well as an overview of a case study; Section 4 shows experimental results to evaluate the proposed strategy; and finally our concluding remarks and further work are discussed in Section 5.

2. Challenges and Related Work

Clusters of workstations are usually managed by software systems named Cluster Resource Managers (CRM) responsible for controlling the access to the cluster machines [4]. This control is based on specification of access rights, scheduling policies and configuration of execution environments. Furthermore, a CRM must provide mechanisms and services to allow the administrators and users to interact to the computing environment, making possible the execution of requests to allocate and release resources, to get information about the allocation queue, access rights and the cluster resources, as well as to submit applications. Some examples of CRMs are Portable Batch System [5], Load Sharing Facility [21], Computing Center Software [15], and CRONO [18].

One key issue in using clusters in computational grids is that, in general, grid users must provide the amount of time and nodes they want to allocate to CRMs. Nevertheless, grid users would rather allocate the maximum computer resources and time they can in order to complete their tasks as soon as possible, which means they usually do not desire to specify allocation constraints. Moreover, if the grid is planetary scale, users do not know the characteristics of all clusters in the grid, making it impossible for them to estimate task duration.

Most CRMs use in their schedulers the backfilling technique [16] which makes possible to fit user requests into gaps of request queues. Therefore, when a scheduler allows users to allocate small partitions, these requests tend to be fulfilled before large requests, thus minimizing the problem of specifying the number of nodes. However, since some CRMs do not allow users to have more than one allocation in the requests queue at same time, it is not possible the allocation of many small partitions at the same time. Another issue that must be considered is that the cluster's administrators do not desire grid users allocating all available cluster resources, since local cluster users may have their applications delayed. Hence, considering these problems, the specification of the number of nodes in grid environments still remains a challenge.

In order to avoid the problems described in this section we propose a strategy that does not rely on a formal cluster allocation to execute grid tasks in clusters. Rather, idle cluster resources are "donated" to the grid manager as soon as they become available (i.e., not scheduled to a user). Thus, in this approach, from the cluster user point of view, there are no grid users in the system, since grid applications are cancelled when local cluster users request resources. Moreover, the utilization rate of cluster nodes tends to increase due to the capacity of exploiting nodes that otherwise would become idle because of external fragmentation.

It is worth noting that our strategy is different from the standard allocation methods used by the available grid resource managers, where allocations must occur, thus generating resource contention for local cluster users. For example, in Condor-G [20], where it is possible to use cluster resources through the Grid Resource Allocation and Management (GRAM) (from Globus Toolkit) [11], a formal allocation takes place, according to the GRAM protocol. In our strategy grid users do not make standard formal allocations to obtain resources to execute their applications. This means that local users will always have preference over grid users, thus when necessary, resources executing grid applications will be preempted. Furthermore, in our approach resources are encapsulated and hence users do not require information such as state of allocation queue, and this difference is important in the context of Grids since users should not know details about the resources available in a site.

In addition, from the perspective of making resources available to users, systems such as SETI@home [2], BOINC [1], XtremWeb [9] and Entropia [6] are also quite

[1] In Master-Worker Applications, if a worker loses its resource, the other tasks still remain performing work and the application overall is not suspended.

Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05)
0-7695-2448-6/05 $20.00 © 2005 IEEE

different. Our strategy does not rely on CPU load, or mouse and keyboard activity. Instead, it uses management information provided by the CRMs to publish the cluster machines to grid users.

3. The Transparent Resource Allocation

The proposed allocation strategy we describe here allows the grid environment to use all idle resources from a cluster without a formal allocation operation. Using this approach, local user applications requesting cluster nodes through standard allocation may preempt remote grid tasks.

Our approach relies on an interaction between CRM and Grid Resource Managers (GRM) to work. We define GRM the software responsible for interacting with grid resources on behalf of grid users. Brokers and Grid Schedulers [17] are examples of GRMs.

The procedures to execute an application in a cluster are not modified by the proposed strategy:

1. Local cluster user requests a node partition to CRM informing the amount of nodes and time. Note that at this stage, remote grid users can also make formal allocations through their GRMs if an interface between their GRMs and the target CRM (e.g., Globus GRAM [12]) is available;
2. Whenever possible, CRM allocates nodes to the requester. Notice that the allocated nodes will not be available for other users during the reserved time;
3. Requester loads an application to the allocated partition;
4. Requester executes the application;
5. Requester releases the allocated partition by calling CRM or the partition is automatically released after the user time expires.

Using our strategy, the procedures to execute grid tasks in a cluster basically consist of:

1. GRM asks CRM to authorize access to idle nodes for a remote grid user – user access rights are verified;
2. GRM submits grid tasks to the target CRM;
3. Grid tasks are loaded into idle nodes;
4. Tasks are executed until they finish or a local cluster allocation needs the nodes being used by the execution of grid tasks.

In the next sections we first provide a description of the steps needed to execute grid tasks on a cluster, and after, we address some implementation issues to put into practice our strategy in a real environment.

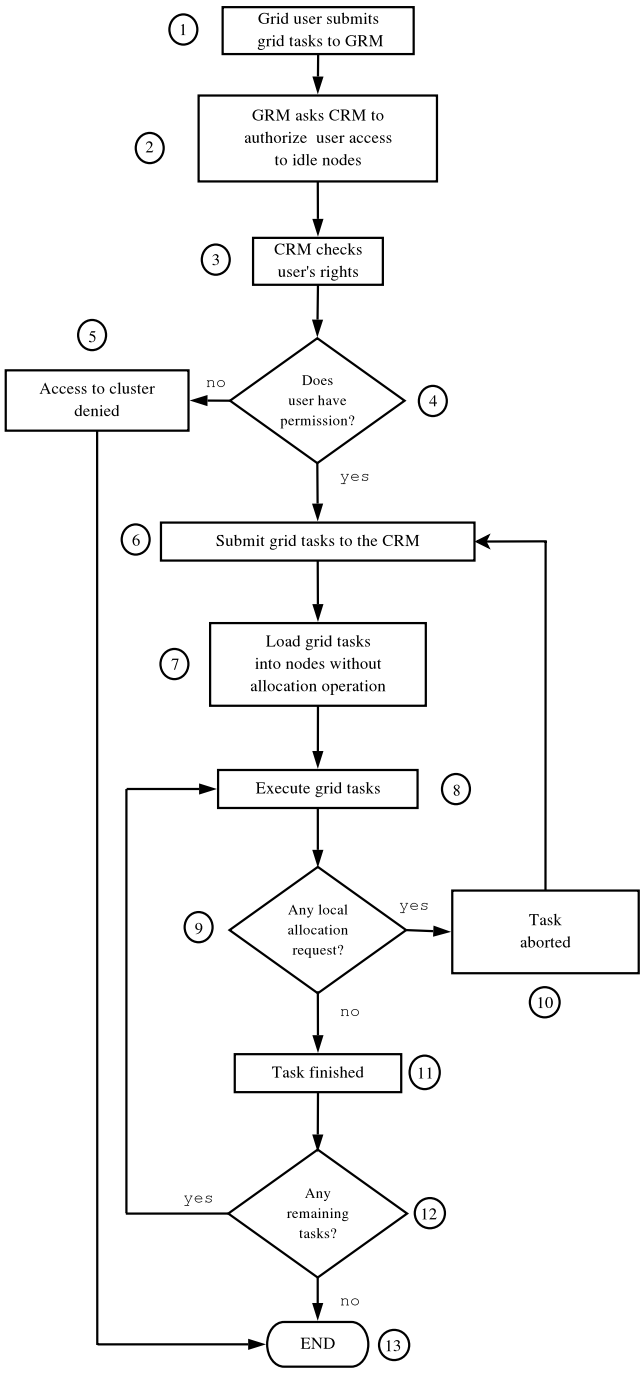

Figure 1. Flowchart representing the execution process of grid tasks on a cluster.

3.1. Detailed description

When a remote grid user needs to execute grid tasks, the user's request is initially handled by GRM (Figure 1).

At Step 1, the grid user submits grid tasks to a computer

Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05)
0-7695-2448-6/05 $20.00 © 2005 IEEE

executing a GRM. At Step 2, GRM asks CRM to authorize the grid user access to any idle cluster nodes. At Step 3, CRM checks user's access rights to receive this access and, at Step 4, it is determined whether one has the appropriate rights. If the user does not have permission to execute tasks, at Step 5 the requested access is denied. Otherwise, if the user has access rights, at Step 6 GRM submits tasks to CRM. At Step 7, those are loaded into the cluster idle nodes, without a formal allocation operation to CRM. At Step 8, grid tasks start the execution.

Note that the number of dispatched tasks increases together with the number of idle nodes. Also, any node on which a task is being executed may be preempted, since the cluster nodes are still available for local cluster users. This step is shown at Step 9. Such a preemption will occur when a local cluster user makes an allocation request of nodes that are idle. If this does not occur for a particular node, the task on that node can continue until it becomes completed (Step 11), and then, if there are tasks waiting to be executed, Step 8 is executed and another task is assigned to that node. Thus, if a task is not preempted, that is, aborted (Step 10), the node continues to process tasks sequentially until there are no tasks waiting to be executed (Step 12), and the procedure finishes at Step 13. Remark that GRM is responsible for monitoring tasks in order to collect results when they finish.

We can observe that the strategy described in this section clearly benefits local cluster users, by preempting grid tasks when resources are requested by the local users. In addition, it is important to mention that if the grid application comprises independent grid tasks, damage to grid users is minimal: cancelling a task only causes such a task to be scheduled again – other tasks still remain executing. In contrast, if the application comprises parallel tasks, the behavior of the application is environment dependent. Note that checkpointing and estimation of resource availability are interesting services to assist in a better execution of such applications. However, in this paper we do not explore these possibilities.

3.2. Identifying idle nodes

In order to put into practice the strategy proposed here in a real computing environment, it is important to deal with the problem of *identifying idle nodes* in a cluster. When a grid user submits tasks to a GRM, this GRM must know the available resources in order to send the user tasks to them. Here we present alternatives to solve this problem. They are presented, respectively, in Figure 2(a), (b) and (c):

1. Implementing an interface between GRM and CRM in which when GRM needs to identify the resources available, it contacts CRM to provide this information. If there are resources available, and the user has access rights, GRM sends the tasks to CRM, and CRM forwards them according to the availability of the idle nodes. Note that CRM also serves as a gateway, since the cluster nodes may not be reached directly by GRM due to network constraints. Both CRM and GRM are able to monitor the execution of the user applications. This basically depends on how the interface is implemented. If a local user requests cluster nodes that are executing grid tasks, CRM could then easily cancel these tasks, since it holds a list of machines being used by grid users.

2. Developing a module to be used as a gateway between GRM and the cluster nodes. A simple method to identify the idle cluster nodes is: when a local allocation starts, CRM executes a pre-processing script/program that provides the list of nodes and who is using each of them for the module. In the same way, a script/program can be executed when a local allocation finishes (post-processing). The pre- and post-processing scripts/programs are usually available in CRMs to set up the user environment and can be easily modified. When the module receives the information about the available nodes, it can execute and cancel grid tasks. Again, both GRM and the module can monitor the grid task executions, this depends on how the interface is implemented. To be able to cancel grid tasks when a cluster machine is requested by local users, the module must keep track of which tasks are executing in each node.

3. Utilizing *node agents*. A node agent is a program executed in each idle node of a cluster. In order to use them, pre- and pos-processing scripts/programs are needed to respectively halting and starting such node agents when machines are allocated and released. Node agents themselves announce their existence to the module, which is responsible to keep track of in which nodes these agents are available. Similar to the second alternative, the main function of the module is to serve as a gateway between GRM and node agents: both GRM and node agents should know network address of the module.

Regarding these three options, the advantage of the first one is that there is no need of another module in the computing environment (Figure 2(a)). CRM becomes responsible for a considerable amount of work, such as forwarding the grid tasks from GRM to cluster nodes, as well as monitoring and cancelling them. The main advantage of the second method is the fact that CRM needs not be adapted (Figure 2(b)). This is an important issue to be considered since it may be difficult to modify the CRM of a site or the site

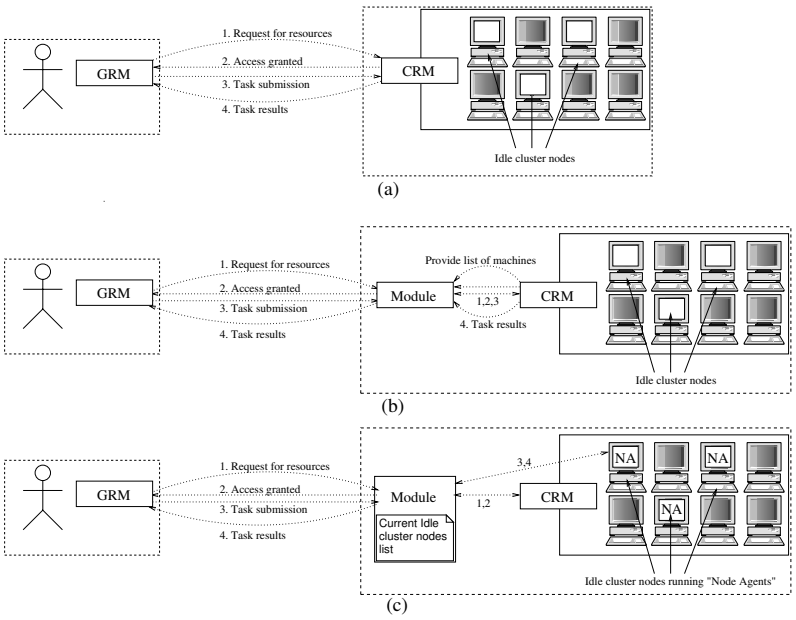

Figure 2. Three ways to implement our approach: (a) Direct communication between GRM and CRM; (b) Addition of a module between GRM and CRM; (c) Use of a module and node agents.

that is providing the resources may not desire to modify its CRM.

The advantage of using node agents is that a grid task can be executed in a sandbox managed by a node agent, thus limiting the capacity of a grid task to damage a cluster node and improving the security of the system (Figure 2(c)). However, the use of node agents can generate additional load into the local cluster network. This load can reduce the performance of the local applications depending basically on the network technology used to connect the front-end machine to the cluster machines, as well as the connection among the cluster machines themselves and the number of cluster machines in the computing environment.

The module described in the previous scenarios is responsible for basically distributing nodes to multiple grid users. However, the strategy used to perform this distribution of resources depends on site policies, which may vary from site to site.

3.3. Case study

The proposed strategy was applied in the OurGrid [3] environment. OurGrid is a free-to-join grid that supports the execution of BoT applications, i.e., those applications composed of a set of independent tasks. Briefly, OurGrid is composed of a user broker (MyGrid) and a meta-scheduler (Peer) which hides all local resources from a site to grid users. Thus, a user needing to use resources to execute an application should only provide a Peer address to MyGrid, instead of all resources addresses. Peers trade resources among themselves so as to maximize local utility using the Networks of Favors [3]. Peers reply resource addresses to the MyGrid broker, and after all, communications are performed directly between MyGrid and site resources. However, due to network policies, it is possible to have a gateway between MyGrid and site resources. To enable cluster utilization with OurGrid, an interface between a Peer and a cluster, called GuMP (Grid Machine Provider), should exist. This interface formally allocates cluster nodes and provides them for the grid.

In order to allow OurGrid to use the resources of our research center[2], which is managed by CRONO scheduler [18], we have implemented the third alternative of the transparent resource allocation (Figure 2(c)). Therefore, *user agents* from OurGrid act as our node agents, the module role has been played by Peer itself, and MyGrid broker acts as GRM. User agents are halted and started by pre- and post-processing scripts available in CRONO.

Using the proposed allocation strategy, the clusters of our research center were deployed to the Pauá community, a Brazilian countrywide grid encompassing 11 universities and research centers [7], as well as a major part of the OurGrid free-to-join grid.

Currently, we are investigating the deployment of our strategy under Globus Toolkit, adopting the approach de-

[2]Research Center in High Performance Computing CPAD-PUCRS/HP - http://www.cpad.pucrs.br.

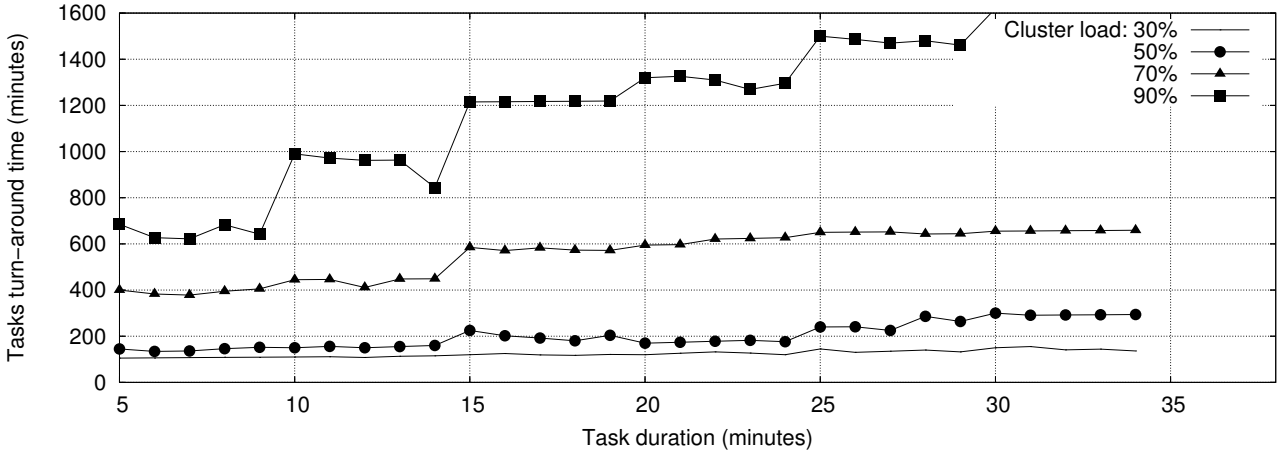

Figure 3. Relation between task duration and turn-around time.

picted in Figure 2(b), in which the module function should be performed by an adapter which would act as a scheduler to interact with GRAM.

4. Experimental Results

In order to evaluate the effectiveness of the proposed strategy, we performed experiments in a simulated environment composed of a cluster with 100 homogeneous nodes that were allocated at different periods of time by local users. The workloads used were based on the utilization rate of our research center main cluster, which varies depending on the period of the year (e.g. beginning of a semester, holiday, etc.). Based on these periods we selected four workloads representing each mean utilization rate: 30%, 50%, 70% and 90%.

Parallel to the cluster workload, we added grid tasks so as to totalize an amount of 7200 minutes of execution in a single cluster node. The grid workload was initially divided in 1440 tasks, each with an execution time of 5 minutes, and then submitted to CRONO, the cluster resource manager used in our experiments. Thus, the turn-around time (time to execute all grid tasks) was measured for each cluster workload.

To investigate the interference of task size in the effectiveness of the proposed strategy, the above experiment was repeated with an increment of one minute in the task execution time. In order to maintain the same experiment conditions, the task execution time was increased and the number of tasks was reduced, thus maintaining the same amount of work (i.e., number of tasks times task duration) to be accomplished. We varied the task execution time from 5 to 34 minutes for each of the four cluster workloads, resulting in 120 scheduling executions.

CRONO processes its request queue with the FIFO algorithm using conservative backfilling [16], resulting in the small grid tasks filling the gaps due to the idle cluster resources. Figure 3 illustrates the turn-around time of the grid workload (y axis) for each task duration (x axis). Each line represents the results for one cluster load. We observe that for cluster low-utilization rates, the influence of task duration in the turn-around time is low. With an increase in the utilization rate, the influence of task duration increases because there are fewer gaps to be filled. Lower task execution times fill the gaps more efficiently, thus resulting in lower turn-around times.

From the turn-around time obtained with the cluster load of 90%, it is possible to conclude that the proposed strategy is less efficient for cluster with high-utilization rates. This is expected since in this case there are fewer gaps to be filled and grid tasks are delayed regardless their duration.

Table 1 shows the turn-around times obtained from the experiments on different environments. In this table, it is possible to notice that for low cluster utilization rates, as expected, the turn-around time of the grid tasks is better than the turn-around time when we have higher cluster utilization rates. We can also observe that this behavior occurs specially for small tasks. For example, 5-minute tasks take 685 minutes in a cluster load rate of 90%, while 30-minute tasks take 1620 minutes, with the same load rate. Remark that both cases have the same amount of work, however, the turn-around time in the second case is more than twice that in the first case.

The utilization of gaps to reduce the turn-around time is illustrated in Figure 4. This figure exemplifies the schedulings for a subset of grid and cluster workloads used in the experiments. In Figure 4 (a), the task duration is shorter than the task duration in Figure 4 (b). As a consequence, the first can exploit the gaps in cluster allocation, reducing its turn-around time. In the later, the duration of gaps was

Table 1. Results of turn-around times on different environments.

Environment	Turn-around time (min)
Dedicated machines	
One cluster node	7200
20 cluster nodes	360
Non-dedicated machines	
5-minute tasks, load rate 30%	105
5-minute tasks, load rate 50%	150
5-minute tasks, load rate 70%	400
5-minute tasks, load rate 90%	685
15-minute tasks, load rate 30%	120
15-minute tasks, load rate 50%	225
15-minute tasks, load rate 70%	585
15-minute tasks, load rate 90%	1215
30-minute tasks, load rate 30%	150
30-minute tasks, load rate 50%	300
30-minute tasks, load rate 70%	655
30-minute tasks, load rate 90%	1620

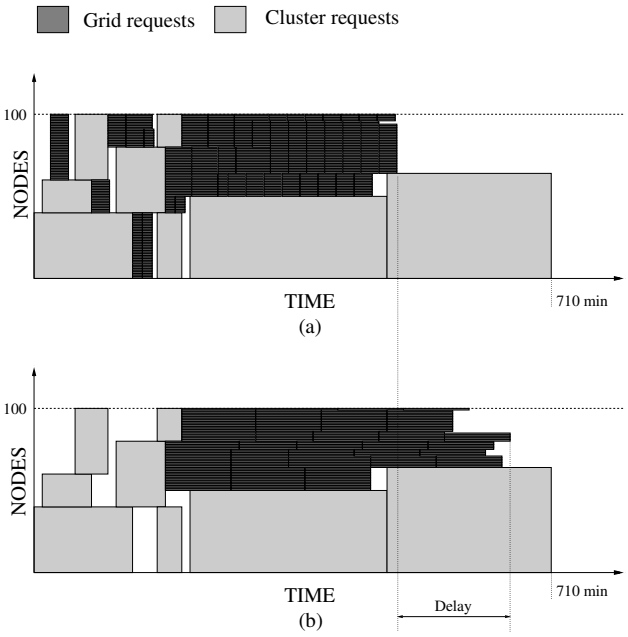

Figure 4. Effects of task duration in turn-around time of grid requests. In (a), grid tasks are shorter, and the gaps among cluster requests can be utilized successfully by them. In (b), these gaps cannot be utilized successfully and the turn-around time increases.

not enough to complete the grid tasks. Thus, these tasks are preempted before they complete, delaying the turn-around time of grid requests.

Based on the data presented in this section, we can conclude that our strategy is particularly useful for applications comprising a large number of short duration tasks to be executed in clusters with medium or low load. Applications composed of medium duration tasks in these clusters should experiment an acceptable turn-around time. For those who need to execute large tasks or need to execute their applications in heavy load clusters, the effort to craft a formal allocation may be worth it.

5. Concluding Remarks and Further Work

Clusters of workstations are high performance computer resources that have been used by scientists to execute complex applications. However, due to the ever increasing complexity of these applications, the scientists currently demand large-scale distributed environments to achieve more processing power. As clusters hold idle resources, mechanisms to explore them are necessary and have been investigated.

In this context, we have proposed a strategy to exploit idle cluster resources for the execution of grid tasks. Our novel approach brings two advantages in contrast to the conventional allocation in grid resource managers: (i) grid requests do not interfere with cluster requests since only idle resources are used and (ii) grid users do not need to make a formal resource allocation to a CRM in order to execute their applications. We believe that the non-interference in the execution time of local cluster applications is important to motivate cluster administrators to publish their resources to grid communities, allowing more researchers to have access to high performance resources.

The experimental executions presented in this paper show a delay in the turn-around time of grid tasks, particularly in high-utilized clusters. Nevertheless, the delay is not significant for highly distributed independent grid tasks and it decreases for shorter tasks. We consider this acceptable since our main goal is to exploit cluster resources to perform grid tasks with no interference in local resource utilization. Our strategy is especially useful in sites that experiment medium and low utilization rates: in such sites, grid users will experiment noticeable performance improvements in relation to a single-machine execution. For sites with high utilization rates, grid users could obtain better results with a traditional formal allocation approach. However, this approach would reduce allocation time of local cluster users.

The proposed strategy has been applied in our research center to provide high performance resources to a real grid

environment, the Brazilian nationwide grid called Pauá. Although the strategy has achieved its purpose, being used in practice, we are still investigating mechanisms to improve it, as well as other techniques to provide cluster resources to grid communities in a transparent way. To exemplify, the use of prediction techniques [14] to estimate resource availability could assist Opportunistic Grids to make better scheduling decisions. Considering, the execution of Master-Worker applications, when the master process loses its machine, the entire application is suspended. Thus, selecting appropriate master machines could prevent the waste of processing time. Selecting groups of machines that have high probability to be idle at particular periods could also simplify the execution of parallel applications, including tightly-coupled parallel applications. Furthermore, based on the resource availability prediction, grid users could better define the frequency that their applications perform checkpointing. Using this strategy, they would also be able to reduce the wasted processing time due to the abrupt loss of resources.

References

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, 2004.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[3] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 61–86, Seattle, 2003. Springer Verlag.

[4] M. Baker, G. Fox, and H. Yau. Cluster Computing Review. Technical report, Northeast Parallel Architectures Center, Syracuse University, 1995.

[5] A. Bayucan. Portable Batch System Administration Guide. *Veridian System*, 2000.

[6] A. A. Chien, S. Marlin, and S. T. Elbert. Resource management in the Entropia system. In J. Nabrzyski, J. M. Schopf, and J. Węglarz, editors, *Grid Resource Management: state of the art and future trends*. Kluwer Academic Publishers, Norwell, 2004.

[7] W. Cirne, F. Brasileiro, L. Costa, D. Paranhos, E. Santos-Neto, N. Andrade, C. D. Rose, T. Ferreto, M. Mowbray, R. Scheer, and J. Jornada. Scheduling in bag-of-task grids: The PAUÁ case. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing*, pages 124–131, Foz do Iguaçu, 2004. IEEE Computer Society Press.

[8] Distributed.net. http://www.distributed.net. April, 2005.

[9] G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb: A generic global computing system. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 582–587, Brisbane, 2001.

[10] FightAIDS@home. http://www.fightaidsathome.org. April, 2005.

[11] I. Foster and C. Kasselman. The Globus project: A status report. In *Proceedings of the Seventh Heterogeneous Computing Workshop*, pages 4–18, Orlando, 1998. IEEE Computer Society Press.

[12] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[13] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, 1999.

[14] K. K. Goswami, M. Devarakonda, and R. K. Iyer. Prediction-based dynamic load-sharing heuristics. *IEEE Transactions on Parallel & Distributed Systems*, 4(6):638–648, 1993.

[15] A. Keller and A. Reinefeld. CCS Resource Management in Networked HPC Systems. *IEEE Computer Society Press*, pages 44–56, 1998.

[16] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel & Distributed Systems*, 12(6):529–543, 2001.

[17] J. Nabrzyski, J. M. Schopf, and J. Węglarz, editors. *Ten Actions When Grid Scheduling*, chapter 2, pages 15–23. Kluwer Academic Publishers, Norwell, 2003.

[18] M. A. S. Netto and C. A. F. De Rose. CRONO: A configurable and easy to maintain resource manager optimized for small and mid-size GNU/Linux cluster. In *Proceedings of the 2003 International Conference on Parallel Processing*, pages 555–562, Kaohsiung, 2003. IEEE Computer Society Press.

[19] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers*, 68(1):91–109, 2003.

[20] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In Fran Berman and Geoffrey Fox and Tony Hey, editor, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., Chichester, 2002.

[21] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23(12):1305–1336, 1993.