Supporting MPI Malleable Applications upon the OAR Resource Manager

Márcia Cristina Cera [1], **Yiannis Georgiou** [2], **Olivier Richard** [2], **Nicolas Maillard**[1], **and Philippe O. A. Navaux** [1]

[1] Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{marcia.cera, nicolas, navaux}@inf.ufrgs.br

[2]Equipe MESCAL – Laboratoire d'Informatique de Grenoble
ZIRST 51, avenue Jean Kuntzmann - 38330 - Montbonnot Saint Martin - France
{Yiannis.Georgiou, Olivier.Richard}@imag.fr

Abstract. *Malleable applications are able to adapt themselves, at execution time, to changes in the amount of resources available. Developing applications with a malleable behavior require some flexibility from programming environment. For instance, MPI-2 provides dynamic processes creation, which can be employed to add some flexibility to MPI applications. Further, an environment that can provide dynamic resources is also required to enable malleability. The OAR resource manager is an open source system composed by high level components, which can be easily extended to integrate new features. Upon this context, this paper exposes how to develop malleable applications using the MPI-2 features, in which the dynamic resources are known through interactions with OAR. Our first results shown that OAR will be able to support malleable jobs and that malleability can provide indeed a better resource utilization with an improvement of almost 35%. Besides resource utilization improvement, we present a discussion about some other advantages brought by some flexibility in job allocation.*

1. Introduction

Malleable applications perform growing and shrinking operations to adapt themselves to changes in resources availability. The development of MPI applications with certain flexibility become possible with the MPI-2 norm, for instance using dynamic process creation [Gropp et al. 1999]. With dynamic processes, when a MPI malleable application performs growing it grants some workload to new resources through processes spawning. In shrinking, processes running on resources announced as unavailable must be stopped and some fault tolerance mechanism is required to avoid crashes. We adopted a simple one: tasks running in resources being shrunk are identified and will be restart in the future. It is not optimal, but ensures application results correctness.

To develop MPI applications with dynamic processes, we designed a scheduler able to determine the physical location of spawning processes [Cera et al. 2006]. Such work was recently upgraded to handle malleable operations take into account resources with dynamic availability. Furthermore, some support in the resource manager is required to provide malleability. OAR [Capit et al. 2005] is a modular and extensible resource manager which we employ to provide malleability. An OAR module was developed to

manage and identify the resources availability, sending such information to the MPI application, which will adapts itself to it. This paper goal is presents the efforts to add some flexibility in job allocation aiming at resources utilization improvements. This work has been developed as part of an international cooperation project between the LIG laboratory and the GPPD/UFRGS (*Grupo de Processamento Paralelo e Distribuído*) group. Many experiences and expertises are changed between both, since the French group focuses in resource management and the Brazilian one in adaptable parallel applications. This paper presents our study case composed by a MPI malleable application, which are scheduled and launched by the OAR resource manager – Section 2. Experimental results are exposed in Section 3. Also, Section 4 discusses some advantages of malleable jobs and, at end, Section 5 concludes this paper.

2. Running MPI Malleable Applications upon OAR

Due to the modularity and flexibility of OAR resource manager, it was possible to construct a prototype, aside the core of OAR, enabling the execution of malleable jobs. The prototype is based on two notions: *(i) Best Effort* job, which is low priority job able to harness the idle resources of a cluster but it can be directly killed when the resource is required; and *(ii)* resource discovery command which provides the current and near-future resources availability. To provide malleable jobs in OAR, we consider that they are composed by a rigid and a *Best Effort* part. The rigid part stays always intact so that it can guarantee the job completeness. In the same time, the *Best Effort* part is responsible for the job flexibility. Hence, using the resource discovery command which informs the application for the variations on resources availability: *Best Effort* jobs can be killed meaning application shrinking or further submitted allowing the application growing.

MPI-2 dynamic process creation [Gropp et al. 1999] can be employed to enable malleability, but some care in application development are demanded. Issues like *when* and *where* spawn processes must be on-line answered. Here, the answer is spawn processes in new resources when they become available. Our previous work provide a scheduler able to determine the physical location of dynamic processes [Cera et al. 2006]. Such scheduler was upgraded to determine the location taken into account the resource availability information provided by OAR. In this way, the scheduler launches malleable operations, which will adapt the application according the resources available. In other words, scheduler performs like a bridge between the MPI application and the OAR.

Our first initiative consider a simple scenario where one malleable application performs upon all idle resources. Extensions of this scenario are awaited in future works. Considering our initial context, the rigid part of the MPI malleable application is the minimum unit required to execute it, *i.e.* one node in our case. The *Best Effort* part is as large as have resources available, which is determined by the resource discovery command. Malleable application starts with all resources answered by the discovery command. When a *Best Effort* job is further submitted characterizing a growing, the scheduler is notified and update its control structures. In the application side, after identify that there are new resources available, it spawns processes charging them. Scheduler will ensure that the spawning processes will be placed in the new resources. In the opposite case, when some nodes are demanded to satisfy arriving jobs, they will required from the *Best Effort* part. The scheduler is notified, performing the shrinking operation and the fault tolerance

procedures. To ensure that this operation will be safe performed, a grace time delay[1] is applied on OAR before the killing of the *Best Effort* jobs [Georgiou et al. 2007].

3. Malleability Improving the Resource Utilization

Aiming to verify the impact of malleable job execution together with rigid ones, we choose a static workload of 5 hours slice of DAS2 workload[2] with 40% cluster utilization. This workload is injected in OAR charging the resources and representing the normal workload of the cluster. At same time one malleable job per time, is submitted and will run upon the free resources, *i.e.* those are not used by the normal workload.

MPI malleable application employed in tests is an implementation of Mandelbrot set, which has adapted to performs growing and shrinking operations. Although Mandelbrot problem do not require a malleable behavior to be solved, we decide use a know MPI application in our tests because our target is the support of malleable operations. The minimum of resources required to start the malleable application is one node and all malleability operations will be performed using whole nodes. In this way, the normal workload jobs are simple 'sleep' jobs just occupying the resources for a specific time, since they do not affect the malleable execution. Results of executing malleable jobs (one per time) are compared to a non-dynamic approach. In other words, the malleable jobs submission is substituted by moldable-besteffort jobs submission. As moldable-besteffort job we define a moldable job, that can be executed upon all the free cluster resources but does not provide flexibility during runtime hence it will be immediately killed (like a besteffort job) when a rigid job asks for resources.

In terms of resources utilization, since the workload makes use of 40% of the cluster, this leaves a 60% of free resources. We observed that moldable-besteffort jobs use a 32% of the idle resources arriving at 72% of total cluster utilization. On the other hand, the malleable jobs use 57% of the idle resources arriving at 97% of overall cluster utilization. Hence, the improvement of the dynamic approach when comparing with the non-dynamic approach is almost **35%** of resources utilization. Furthermore, we observed the number of jobs executed in 5 hours of experimentation. In the dynamic context we obtain 8 successfully Terminated malleable jobs compared to 4 Terminated and 5 in Error State for the non-dynamic context. Finally the impact of the response time for the normal workload was also measured and the result was 8 sec of average response time in case of non-dynamic context, compared to 44 sec for the dynamic one. The response time for malleable MPI approach is explained by the grace time delay to ensure safe shrinking operations, which was 40 sec. Table 1 resumes the results exposed.

Table 1. Comparison between malleable and moldable-besteffort jobs.

	idle resources used	jobs Terminated	jobs Error	response time
Malleable	57%	8	0	44 sec
Moldable-besteffort	32%	4	5	8 sec

[1]Amount of the time that the OAR system waits before destinate the resources to another job, ensuring that they are free.

[2]http://www.cs.huji.ac.il/labs/parallel/workload/l_das2/index.html

4. Discussion about Malleability Advantages

Previously, we exposed the gain in resource utilization brought by the execution of malleable jobs (one per time) and rigid ones. Kalé et al. [Kalé et al. 2000] exposes some other advantages of malleable jobs. In this section, we discuss these advantages and make a relation with OAR and MPI malleable application context.

- Resize low priority jobs to respond to higher priority ones, insted of preempt them. In Table 1, malleable jobs (low priority) are resized to supply the arriving rigid ones and always finish their execution successfully. On the other side, moldable-besteffort jobs are sometimes preempted and get Error Status;
- Use idle-cycle jobs. MPI malleable jobs performs in the gaps of cluster resources enabling the utilization of idle-cycles. As shows in Table 1, the idle-cycle utilization of malleable MPI jobs is bigger than the moldable-besteffort ones;
- Improve job efficiency by its shrinking. Applications with a poor scalability usually are inefficient with many resources. In these cases, destinate some resources to other applications can improve their performance. This is not the case of Mandelbrot set that has a trivial parallelization. But this feature can taken into account in OAR requiring adaptations in the scheduling policies;
- Small-duration jobs can be promptly responded by shrinking malleable jobs. OAR can shrinking malleable jobs to attend small jobs reducing their waiting time, and after growing again when the small jobs finishes.

5. Conclusion

This paper shown a malleable application that take advantage of MPI-2 dynamic features. Such application, helped by a dynamic processes scheduler, execute upon the unused resources of a cluster improving its utilization. The malleable applications are launched by OAR resource manager, which is responsible to manage the resources dynamicity. OAR and the scheduler interact exchanging informations about the resources availability. In this paper, we have shown that malleable jobs upon OAR can improve the resource utilization of almost 35%. This work is part of a cooperation project between LIG, research about OAR, and GPPD/UFRGS, research about MPI malleability.

References

Capit, N. et al. (2005). A batch scheduler with high level components. In *5th International Symposium on Cluster Computing and the Grid*, pages 776–783. IEEE Comp. Society.

Cera, M. et al. (2006). Improving the dynamic creation of processes in mpi-2. In *13th European PVMMPI Users Group Meeting*, volume 4192 of *LNCS*, pages 247–255.

Georgiou, Y. et al. (2007). Evaluations of the lightweight grid cigri upon the grid5000 platform. In *3th IEEE Int. Conf. on e-Science and Grid Computing*, pages 279–286.

Gropp, W., Lusk, E., and Thakur, R. (1999). *Using MPI-2 Advanced Features of the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, USA.

Kalé, L. V., Kumar, S., and DeSouza, J. (2000). An adaptive job scheduler for time-shared parallel machines. Technical Report 00-02, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign.