Toward MPI Malleable Applications upon a Scientific Grid Environment

Márcia Cristina Cera, Nicolas Maillard and Philippe O. A. Navaux
Instituto de Informática - UFRGS
Grupo de Processamento Paralelo e Distribuído
{*marcia.cera, nicolas, navaux*}*@inf.ufrgs.br*

Abstract

Some kind of dynamicity is eventually reached in nowadays' parallel architectures. Such dynamicity can be derived from grid features as well as from shared utilization of the cores in multicore machines. In such context, parallel applications must adapt themselves to availability changes to ensure good resource utilization. Malleable applications provide such adaptability and their support upon a scientific grid environment is our research target. Indeed, the resource management systems must be able to treat and manage dynamic resources. This paper describes the requirements and challenges to enable malleable application upon a grid scientific environment. The work's focus is the interactions required between the parallel application and the resource management system to enable malleability. Our case study is MPI-2 malleable applications upon the OAR resource manager. Our work's contribution is about the improvement in resource utilization executing malleable applications when compared to executions of non-dynamic applications.

1. Introduction

Today's parallel architectures eventually include some dynamicity or flexibility in their resource's availability. For instance, dynamicity can be a feature of a parallel architecture like grids [9], in which resources become available and unavailable constantly. Another option is consider the shared utilization of multicore machines. As applications have different execution times, the cores' availability change dynamically. In this context, parallel applications running over today's parallel architectures need to be flexible enough to cope with variation in resources provided by the underlying architectures. Malleable applications are flexible parallel applications which are able to increase (growing operation) and decrease (shrinking operation) the amount of resources used at runtime.

MPI (Message Passing Interface) [11] is a well-known communication library in high performance computing area. There are many benchmarks (for instance, Linpack [6] and NAS [5]) and Great Challenges applications such as weather, astrophysics, quantum chemistry, nuclear simulations, and so on [2], implemented with MPI. Further, most of current MPI distributions already includes MPI-2 [12] features such as dynamic process creation, among others. Dynamic process creation, also called dynamic processes, enable the development of flexible MPI applications. In MPI-2 context, a malleable application can be seen as an application that uses dynamic processes to load new resources (growing operation) and some fault tolerance mechanism to safely end processes in resources becoming unavailable (shrinking operation).

Beside issues about dynamicity in parallel application programming, dynamicity also must be treated in the resource management context. According Feitelson et al. [8] a good approach towards the support of malleable jobs can be achieved by a co-design of runtime-system and programming environment. Our work also consider the issues about the co-design between the programming environment (MPI-2 used to provide malleability) and the resource management. The treatment of dynamic resources, *i.e.* resources with a dynamic availability, from resource management systems is a challenge. We investigate the requirements of such treatment considering a generic resource management system, the OAR [3]. It is an open source manager, which can be easily extended to integrate new features thanks its open architectural choices based on high level components (Database and Perl/Ruby Programming Languages). The OAR malleability support has been developed in association with the French LIG (*Laboratoire d'Informatique de Grenoble*) laboratory.

This paper describe the requirements of both, programming environment and resource management systems, to provide adaptation to resource's dynamicity. We focus our work in MPI-2 malleable applications and in the OAR resource manager. The remain of this pa-

per shows an overview about malleability and dynamic resources support in Section 2. In the following, Section 3 presents our first consideration about how to provide malleability in OAR. Section 4 exposes the adaptations required in the application programming side to enable malleability, *i.e.* dynamic resource adaptation. Our first results with malleable applications upon a production cluster are shown in Section 5. Finaly, the Section 6 concludes this paper.

2. Overview about Malleability Support

There are relevant theoretical results describing requirements, execution scheme and algorithms to efficiently schedule malleable applications [19]. Moreover, there is a growing interest to offer malleability in practice, because it can improve both utilization and response time [8, 18, 17, 21, 15, 1, 7]. Observing the related works, most of the initiatives aiming at malleability provides interactions between resource management systems and the applications (reflecting in programming environment). In this way, malleable applications can execute taking into account the resource availability information that is provided by the resource manager. For instance, Dynaco [1] and PCM (Process Checkpointing and Migration) [20, 7], enable dynamic resources adaptation to MPI applications, in which both consider the availability information received from their specific own resource manager systems.

Furthermore, some works focus in the improvement quality of service (QoS) metrics, such as resource utilization and response time. Initiatives like AMPI (Adaptive MPI) [13, 14] and APM (Application Parallelism Manager) [16] are examples and both use resource availability information to improve their QoS metrics. Other works are concerned about provide malleability in a way completely independent of the application programming environment. For instance, Utrera et al. [21] combines moldability (to decide the amount of processes at start time), folding (to change the amount of resources used at running time) and Co-Scheduling (to share resources) to enable better resource utilization by MPI applications. Folding handles cores transparently during the application execution. As a consequence, the initiative provides better resource utilization and reduces response time. However, it also uses resources availability information to take the folding decisions.

3. Supporting Dynamic Resources in OAR

Previous section highlights initiatives with co-design between programming environment and the resource management to provide malleability. We also develop a co-designed solution aiming at execute MPI-2 malleable applications through the OAR resource manager. The current section describes the OAR requirements to support malleable jobs.

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 OAR features: *(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.

In OAR, malleable jobs are composed by a rigid and a *Best Effort* part. The rigid part never change its size which is the minimum amount of resources required by the application, so it can ensure job completion. On the other hand, 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: existing *Best Effort* jobs can be killed, shrinking the application, and new such jobs can be submitted, causing the application grow.

4. Towards MPI Malleable Applications

MPI-2 dynamic process creation [12] (or simply dynamic processes), can be employed to enable malleability. Thus, when a MPI malleable application performs a growing operation, 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.

The employment of dynamic processes demands some care in application development. Issues such as *when* and *where* to spawn processes must be on-line answered. In the target context, the answer is to spawn processes in new resources when they become available. Our previous work provides a dynamic processes scheduler able to determine the physical location of dynamic processes [4]. 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 availability. In other words, scheduler performs like a bridge between the MPI application and the OAR.

Our first initiative considers 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 OAR malleable job is the minimum unit required to execute the MPI

application, *i.e.* one node in current case. The *Best Effort* part is as large as the resources available, which is determined by the resource discovery command. One malleable application starts with all resources answered by the discovery command.

Growing operations are characterized by a further submission or a *Best Effort* job. When this occurs, the dynamic processes scheduler is notified and updates its control structures. In the application side, after identifying that there are new resources available, it spawns processes charging them. Dynamic processes 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 dynamic processes 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 is applied on OAR before the killing of the *Best Effort* jobs [10]. Grace time delay represents the amount of the time that the OAR system waits before destinate the resources to another job, ensuring that they are free.

5. Initial Results and Perspectives

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[1] 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 at a time is submitted and will run upon all free resources, *i.e.* those are not used by the normal workload.

The MPI malleable application employed in tests is an implementation of Mandelbrot set, which was adapted to performs growing and shrinking operations. Although Mandelbrot problem does not require a malleable behavior to be solved, we decide use a known MPI application in our tests. As our target is study the requirement of malleable operations, our Mandelbrot application, which has the desired behavior, is enough. The minimum of resources required to start the malleable application is one node and all malleability operations will be performed using whole nodes, according to the MPI distribution restriction used in our tests. The normal workload jobs (from DAS2 workload) 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 such non-dynamic approach, malleable jobs submissions are substituted by moldable-besteffort jobs submissions. As moldable-besteffort job we define a moldable job that can be executed upon all the free cluster resources but does not provide flexibility at runtime, hence it will be immediately killed (like a *Best Effort* 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 60% of free resources. We observed that moldable-besteffort jobs use 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 malleability when comparing with the non-dynamic approach is greater than **25%** in 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 such results.

	Malleable	Moldable-besteffort
Idle resources used	57%	32%
Jobs Terminated	8	4
Jobs Error	0	5
Tesponse time	44 sec	8 sec

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

Results exposed in Table 1 confirm that the flexibility of malleable applications can improve the QoS metrics of scientific grid environment. Also, we identify that the complexity behind the support of this flexibility is compensated by the gain achieved. Such initial experiments shows that the effort to adequate the OAR and the MPI applications to malleable features are viable, and thus meet their goal. From now, our efforts aiming at a calibration of the interactions between OAR and dynamic processes scheduler, as well as the standardization of the communication between them. Furthermore, we desire to stablish an interface to simplify the development of MPI-2 malleable application, which will enable testing over a broader set of applications.

6. Final Remarks

This paper shown the requirement to support MPI-2 malleable application upon the OAR resource manager. MPI-

1 http://www.cs.huji.ac.il/labs/parallel/workload/l_das2/index.html

2 malleable application, helped by a dynamic processes scheduler, executes upon the unused resources of a cluster, improving its utilization. Such application is launched by OAR resource manager, which is responsible to manage the resource's dynamicity. OAR and the dynamic processes scheduler interact exchanging informations about the resources availability. Our initial experiments associate flexible jobs with the normal/rigid ones aiming improve the resources utilization. We identify an improvement of almost 35% from malleable jobs when compared with a non-dynamic approach. Such result shows that the complexity behind the malleability support is outperformed by the gain.

Our perspective on future works includes the analysis of multiple malleable jobs behavior executing together with rigid ones. The execution of multiple malleable jobs at the same time depends of the scheduling policy adopted by OAR. Such policies are responsible for defining when and with how many resources each malleable job will start. In this way, our future works also foresee scheduling policy issues. In addiction, some improvements in the standard used to communicate the OAR with the dynamic processes scheduler is considered in our future works.

Acknowledgements

The author would like to thank CAPES to support this research. Further, an important support was provided by the French LIG laboratory through the MESCAL project and the Grid5000 experimental testbed. Also, we thank the researchers Olivier Richard and Yiannis Georgious for all discussions and contributions to this work.

References

[1] J. Buisson, F. André, and J.-L. Pazat. Supporting adaptable applications in grid resource management systems. In *8th IEEE/ACM International Conference on Grid Computing*, pages 58–65. IEEE, 2007.

[2] R. Buyya. *High Performance Cluster Computing: Programming and Applications*. Prentice Hall, NJ, USA, 1999.

[3] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *5th International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, pages 776–783, Cardiff, UK, 2005. IEEE Computer Society.

[4] M. C. Cera, G. P. Pezzi, E. N. Mathias, N. Maillard, and P. O. A. Navaux. Improving the Dynamic Creation of Processes in MPI-2. In *LNCS - 13th European PVMMPI Users Group Meeting*, volume 4192/2006, pages 247–255, Bonn, Germany, 2006. Springer Berlin / Heidelberg.

[5] D. Bailey et al. The NAS parallel benchmarks. Technical Report RNR-91-002, NAS Systems Division, Jan. 1991.

[6] J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[7] K. El Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. Malleable iterative mpi applications. *Concurr. Comput. : Pract. Exper.*, 21(3):393–413, 2009.

[8] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer-Verlag, 1996.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 2003.

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

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Massachusetts, USA, Oct. 1994.

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

[13] C. Huang, O. S. Lawlor, and L. V. Kalé. Adaptive mpi. In *Languages and Compilers for Parallel Computing, 16th International Workshop, LCPC 2003*, Lecture Notes in Computer Science, pages 306–322, College Station, TX, USA, 2003. Springer.

[14] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé. Performance evaluation of adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[15] J. Hungershföer, A. Streit, and J.-M. Wierum. Efficient resource management for malleable applications. Technical report, 2001.

[16] J. Hungershöfer. Increased scheduling quality by utilizing the flexibility of malleable jobs. In *17th International Conference on Parallel and Distributed Computing Systems*, pages 72–77, 2004.

[17] J. Hungershöfer. On the combined scheduling of malleable and rigid jobs. In *16th Symposium on Computer Architecture and High Performance Computing*, pages 206–213, 2004.

[18] L. V. Kalé, S. Kumar, and J. DeSouza. A malleable-job system for timeshared parallel machines. In *2nd Int. Symposium on Cluster Computing and the Grid*, page 230. IEEE, 2002.

[19] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci.*, 13(4):613–627, 2002.

[20] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. Dynamic malleability in iterative MPI applications. In *Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 591–598. IEEE, 2007.

[21] G. Utrera, J. Corbalán, and J. Labarta. Implementing malleability on mpi jobs. In *IEEE 13th International Conference on Parallel Architectures and Compilation Techniques (PACT 2004), Antibes Juan-les-Pins, France*, pages 215–224. IEEE Computer Society, 2004.