Morco: Middleware Framework for Long-running Multi-component Applications on Batch Grids *

Sivagama Sundari M
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
sundari@rishi.serc.iisc.ernet.in

Sathish S Vadhiyar
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
vss@serc.iisc.ernet.in

Ravi S Nanjundiah
Centre for Atmospheric & Oceanic Sciences
Indian Institute of Science, Bangalore, India
ravi@caos.iisc.ernet.in

ABSTRACT

While computational grids with multiple batch systems (batch grids) have been used for efficient executions of loosely-coupled and workflow-based parallel applications, they can also be powerful infrastructures for executing long-running multi-component parallel applications. In this paper, we have constructed a generic middleware framework for executing long-running multi-component applications with execution times much greater than execution time limits of batch queues. Our framework coordinates the distribution, execution, migration and restart of the components of the application on the multiple queues, where the component jobs of the different queues can have different queue waiting and startup times. We have used our framework with a foremost long-running multi-component application for climate modeling, the Community Climate System Model (CCSM). We have performed real multiple-site CCSM runs for 6.5 days of wallclock time spanning three sites with four queues and emulated external workloads. Our experiments indicate that multi-site executions can lead to good throughput of application execution.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed applications*; D.2.11 [**SOFTWARE-ENGINEERING**]: Software Architectures; J.2 [**PHYSICAL SCIENCES AND ENGINEERING**]: Computer Applications—*Earth and atmospheric sciences*

General Terms

Design,Documentation,Performance

Keywords

multi-component applications, climate models, batch systems, checkpointing, migration, rescheduling

*This work is supported partly by Ministry of Information Technology, India, project ref no. DIT/R&D/C-DAC/2(10)/2006 DT.30/04/07 and partly by Department of Science and Technology, India. project ref no. SR/S3/EECE/59/2005/8.6.06

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

1. INTRODUCTION

Computational grids have been used over the years for efficient execution of different kinds of parallel applications, including loosely-coupled, workflow-based and in some cases tightly-coupled applications [1, 5, 6]. Various robust grid middleware frameworks have been developed for supporting grid executions of these kinds of applications [7, 10]. However, grids have not been sufficiently employed for executions of long-running multi-component applications (MCAs).

In this paper, we have developed **Morco** (**M**iddleware framework for l**o**ng-**r**unning multi-**co**mponent applications), a framework for execution of long-running multi-component applications (MCAs) on multiple batch systems of a batch grid. The long-running applications span multiple execution time limits of the batch queues and hence span multiple submissions of the individual components to the queues of the systems. **Morco** automatically coordinates the execution of the components on the different queues. We have employed a novel execution model in which the set of active batch systems available for execution is dynamically shrunk and expanded, and the components are rescheduled on possibly different batch systems. This is illustrated in Figure 1 with an example of a two-component application executing on two batch queues. We have used our framework for execution of a foremost long-running multi-component parallel application, CCSM (Community Climate System Model) [3, 4] with four components across four queues.

2. MORCO - GRID MIDDLEWARE FRAMEWORK

Our Morco middleware framework consists of three primary components to synchronize the executions of the components of a multi-component application (MCA) on multiple batch systems: a *coordinator* that determines mapping of components, and schedules and reschedules the component executions on the systems, a *job monitor* on each front end node of the batch systems that interfaces with the coordinator, and a *job submitter* on the front end node that repeatedly submits a MCA job upon completion of the previous MCA job. Our framework also consists of a *MCA job script* which executes and re-executes the MPI multi-component application on a system corresponding to specified mappings of components to processors at various points of time within a MCA job submitted by the *job submitter*. In addition to these components, our framework also includes some components to ensure fault-tolerance. The

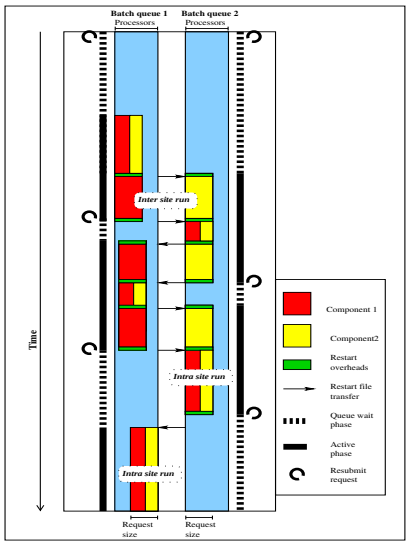

Figure 1: Execution of a long-running two-component application on two sites

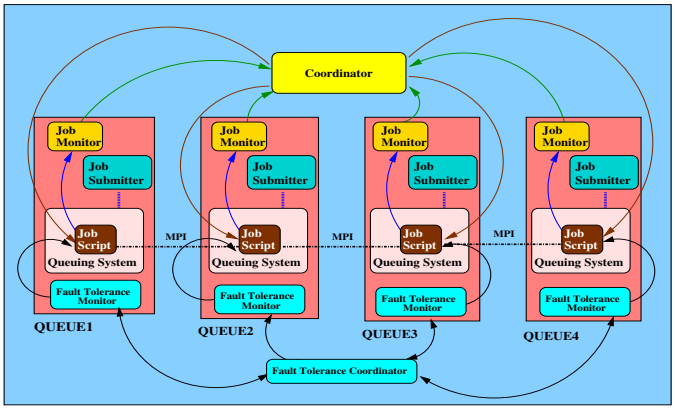

Figure 2: Morco - Grid Middleware Framework

architecture is illustrated in Figure 2.

An application job is submitted to each of the batch systems with a request for a specific number of processors by the Job Submitter. When a job on a batch system is active (not waiting in queue), it coordinates with our middleware framework and executes some components of the application depending upon the number of active batch systems at that instant. The components executed by the job can change when the number of active batch systems changes. When the job is close to its maximum execution time-limit on the batch system, it coordinates with the rest of our framework, creates the necessary restart data and exits the queue. The job submitter submits a new job after the job exits the queue.

When a job submitted to a system becomes active or has entered the execution state after waiting in the batch queue, the job monitor on the system informs the coordinator of the *START* status of the job. Similarly, when the batch job on one of the active systems is about to reach the execution time limit of the system, the job monitor at the system sends a *STOP* message to the coordinator. The coordinator sends stop signals to the MPI jobs executing on all active batch systems. The MPI jobs, after receiving the stop signals, create the restart files and stop executions. The job monitors at each site then send a *STOPPED* message to the coordinator. The coordinator waits for the *STOPPED* message from all the previously active batch systems.

Based on the number of active batch systems, the coordinator then uses a genetic algorithm to determine the schedule of execution of the multi-component application on the set of active batch systems. The schedule contains the set of components and the number of processors for the components allocated to each active system. The schedule is sent to the job monitors of the active systems which write the schedule to files called *component-config* files. It also transfers the *restart dump* files generated by the applications in the previous set of active systems to the new schedule, and takes a backup of the restart files for use in case of a complete system failure, thereby providing fault-tolerance. It then informs the batch jobs of the active systems to resume execution. The batch job of each active system reads its *component-config* and executes its set of components on the set of its processors as specified in the *component-config* file.

Some of the salient features of our Morco framework are listed below:

1) **Long-running multi-component applications:** Since the applications considered are long-running, it is reasonable to assume that they can be stopped and restarted either using in-built restart facilities or using an external checkpointing library. Since components of most coupled multi-component applications cannot be split, we perform distribution of the communicating components across the sites rather than splitting a component across sites.

2) **Multiple queues across multiple sites:** The framework can handle variable number of queues and clusters at different sites. It can handle multiple queues within the same cluster as well as queues across clusters. Its primary function is to coordinate execution of a single large long-running multi-component application through jobs submitted multiple times to each of the multiple queues.

3) **Dynamic reconfiguration of application:** Our framework supports multiple reconfigurations of application within a submission corresponding to other submissions becoming active or inactive. It automatically detects the job submitted to queue becoming active and reconfigures the multi-component application to include the newly available resources. Similarly, it also automatically detects when a job in a queue is close to timeout and reconfigures the multi-component application restricting it to other active queues. It uses a genetic algorithm based scheduler to dynamically compute the configuration, i.e. the mapping of components to processors. We model each chromosome as a string of processor-sizes of components, and evaluate it with an *application-specific fitness function*, which gives the best execution rate possible with these component sizes across all possible mappings on the current set of active queues. To estimate the execution rates, we have developed a performance model for predicting performance of the multi-component application across multiple sites.

4) **Portability to Different Batch Systems:** At any instant there is exactly one job corresponding to our target application in each queue, and hence it does not unfairly affect the queue wait times of other external jobs in the queue. Our framework performs and monitors our submission and acts in response to the actions of the schedulers in the batch system; thus, it can be used without any modifications to existing independently managed batch schedulers at various sites. Also, since its only interaction with the batch sys-

tem is to submit a job, it can be used with a wide range of batch system schedulers.

5) **Fault-tolerance:** Since our framework supports execution of an application across an open network with potential network instabilities that can cause failures in MPI executions, we have included an adjunct fault-tolerance framework for automatically detecting such and other failures and re-running the application from the previous restart-dumps. We have also incorporated a large-scale fault-tolerance feature within the coordinator daemon of the main framework to handle major failures such as node failures resulting in failure of the framework daemons.

3. EXPERIMENTS AND RESULTS

We tested our Morco middleware framework by executing CCSM across four batch queues in three clusters, namely, *fire-16*, a AMD Opteron cluster with 8 dual-core 2.21 GHz processors, *fire-48*, another AMD Opteron cluster with 12x2 dual-core 2.64 GHz processors, and *varun*, an Intel Xeon cluster with 13 8-core 2.66 GHz processors. Four queues were configured on these systems with OpenPBS: one queue, *queue-14*, of size 14 on *fire-16*, one queue, *queue-48*, of size 48 on *fire-48*, two queues, *queue-32* and *queue-64*, of sizes 32 and 64, respectively, on *varun*. The AMD clusters are located at the Supercomputer Education and Research Centre and the Intel Xeon cluster is located at the Centre for Atmospheric and Oceanic Sciences, and are connected through a campus network with a bandwidth of around 500 Kbps. The AMD clusters are connected to each other with Gigabit ethernet switches. The connections within the three clusters are using switched Gigabit Ethernet.

External loads were simulated by submitting synthetic MPI jobs to the queuing systems based on the workload model developed by Lublin and Feitelson [8]. The maximum execution time limit for all jobs on all queues was set to 12 hours. The coordinator was started on the front-end node on *fire-16*. A job monitor and a job submitter corresponding to each queue were started on the front-end of its cluster.

We performed a long-running experiment in which our Morco framework executed CCSM for a period of 6.5 days across the 4 queues on 3 systems during which climate of 5 years, 4 months and 26 days was simulated. As the jobs on each of the four queues became active and inactive, the CCSM runs were automatically reconfigured and restarted by our framework. The execution profile of CCSM on the various queues during this multi-site execution is shown in Figure 3.

The figure shows the location of execution of various CCSM components along the execution time-line as the configurations change. The figure comprises of four subplots corresponding to the four queues in our experiment, as indicated by the labels at their top right corners. The x-axis shows the experiment timeline in hours, while the y-axis has the total number of processors available in each queue. The colored regions correspond to the execution of CCSM, while the white regions correspond to processor-periods that are either unused or used by other jobs in the queue. Each color in the figure corresponds to a single component. For any given x-axis value corresponding to a given time instant, the components executing in each queue and the number of processes used by each component are indicated by the component-colors and the height of each color, respectively. For example, during the 2^{nd}-12^{th} hour of execution, the land component, represented by the green bar, executed on *queue-14* and the atmosphere component, represented by the dark blue bar, executed on *queue-48*. The experiment involved a total number of 35 reconfigurations of CCSM components performed automatically by our framework.

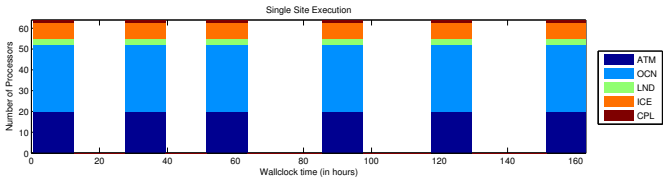

Figure 4: Single-site Execution Profile

As the variations in the heights of colored regions in the figure indicate, the component sizes and hence the total number of processes used for CCSM execution varies with the set of active queues. Also, there are noticeable periods of inactive systems in each queue indicated by the white regions. Our framework automatically handles large changes in the number of active queues and the number of processors. For example, in the 1^{st} hour, only *queue-14* was active and all the five CCSM components were executed in this queue. This is indicated at hour 1 on the x-axis by the presence of all four colors in the topmost subplot and the blank regions on other subplots. The figure also shows two phases during the experiment near the 100^{th} and 130^{th} hours, during which there were zero active systems, i.e., when the CCSM jobs on all the batch systems were waiting in the queue. The black vertical lines indicate points during our experiment when there were system failures due to power failures resulting in the shutdown of the entire system on one of the sites. Our fault-tolerant framework takes backups of all the component restart files at each CCSM reconfiguration. We used these restart backups to continue our experiment from these points.

Hence, as illustrated, whenever new batch systems become active or active systems reach execution time limit, our Morco framework automatically stops the execution on the current configuration, calculates a new configuration with different component sizes and different locations, reconfigures the CCSM components to the new configuration and continues the execution. It also handles smoothly the cases of no active queues and all queues becoming active. Thus, the experiment has demonstrated that our middleware framework can be effectively used for robust long-running simulations.

A similar execution profile for a single-site run is shown in Figure 4. As shown in the figure, there are larger gaps between two CCSM executions when CCSM is executed only on a single site than when it is executed across multiple sites using our Morco framework as shown in Figure 3. The total length of the gaps is 88 hours for single-site runs and only 25 hours for multi-site runs. Thus, multi-site executions of CCSM using our Morco framework ensure continuous progress and regular updates of long-running climate simulations.

Figure 5 compares the execution progress of CCSM on multi-site runs with that on single-site runs. Each point in the figure corresponds to a restart point in the experiments. The almost flat-regions of the multi-site execution curve between 120-140 hours is due to all batch jobs becoming inactive during this time as seen in Figure 3. The other flat regions (e.g. 40-60 hours) correspond to execution on small number of processors on *queue-14*. We find that the progress of execution with multiple sites is comparable with

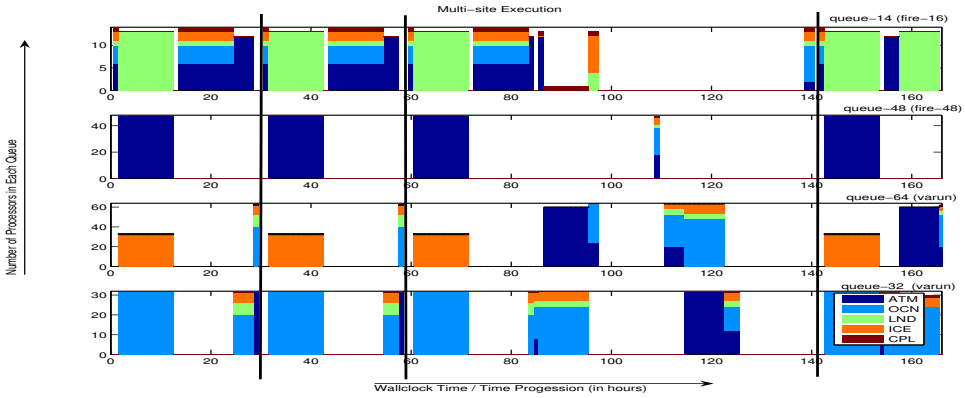

Figure 3: Multi-site Execution Profile

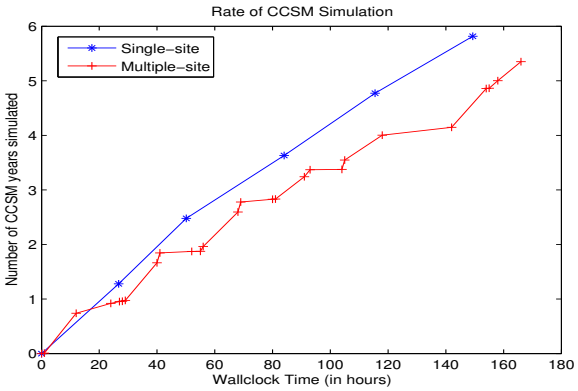

Figure 5: Rate of CCSM Simulation

single-site executions, in spite of the various overheads related to multi-site executions including system failures due to power fluctuations, restart overheads, multiple reconfiguration and rebuilding overheads.

4. RELATED WORK

Buisson et al. [2] in their work on scheduling malleable applications in multi-cluster systems, have developed a middleware framework called DYNACO for their application runner, MRunner, to execute malleable applications. They support dynamically growing and shrinking the applications processor set. Our framework, though similar, is designed for multi-submission executions on generic batch scheduling systems. Markatchev et al. [9] have developed a middleware framework for checkpointing, migration and reconfiguration for execution of traditional long-running applications. They also consider batch systems and execution time limits of the systems, and perform migration of batch jobs before reaching the time limits. However, unlike our framework, their work does not support execution of an application job co-allocated across multiple batch systems. Thus our work is unique in building a framework for co-allocation to execute long running applications spanning multiple batch submissions with the batch queues of potentially different existing job execution policies and where the number of batch queues available for execution can change during the execution of the application.

5. REFERENCES

[1] A. Bhatele, S. Kumar, M. Chao, J. Phillips, Z. Gengbin, and L. Kale. Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms. In *IPDPS '08: Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2008.

[2] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, and D. Epema. Scheduling Malleable Applications in Multicluster Systems. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 372–381, 2007.

[3] Community Climate System Model (CCSM). `http://www.ccsm.ucar.edu`.

[4] W. Collins, C. Bitz, M. Blackmon, G. Bonan, C. Bretherton, J. Carton, P. Chang, S. Doney, J. Hack, T. Henderson, J. Kiehl, W. Large, D. McKenna, B. Santer, and R. Smith. The community climate system model: Ccsm3. 1998.

[5] B. Howe, P. Lawson, R. Bellinger, E. Anderson, E. Santos, J. Freire, C. Scheidegger, A. Baptista, and C. Silva. End-to-End eScience: Integrating Workflow, Query, Visualization, and Provenance at an Ocean Observatory. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 127–134, 2008.

[6] Y. Joshi and S. Vadhiyar. Analysis of DNA Sequence Transformations on Grids. *Journal of Parallel and Distributed Computing*, 69(1):80–90, 2009.

[7] S. V. Kumar, P. Sadayappan, G. Mehta, K. Vahi, E. Deelman, V. Ratnakar, J. Kim, Y. Gil, M. Hall, T. Kurc, and J. Saltz. An Integrated Framework for Performance-based Optimization of Scientific Workflows. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 177–186, 2009.

[8] U. Lublin and D. Feitelson. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[9] N. Markatchev, C. Kiddle, and R. Simmonds. A Framework for Executing Long Running Jobs in Grid Environments. In *HPCS '08: Proceedings of the 22nd International Symposium on High Performance Computing Systems and Applications*, pages 69–75, 2008.

[10] K. Nomura, R. Seymour, W. Weiqiang, H. Dursun, R. Kalia, A. Nakano, P. Vashishta, F. Shimojo, and L. Yang. A Metascalable Computing Framework for Large Spatiotemporal-scale Atomistic Simulations. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–10, 2009.