Middleware for Long-Running Applications on Batch Grids

Sivagama Sundari M*, Sathish S Vadhiyar*, Ravi S Nanjundiah†
*Supercomputer Education and Research Centre
†Centre for Atmospheric & Oceanic Sciences
Indian Institute of Science, Bangalore, India
sundari@rishi.serc.iisc.ernet.in, vss@serc.iisc.ernet.in, ravi@caos.iisc.ernet.in

Abstract—**Computational grids with multiple batch systems (batch grids) can be powerful infrastructures for executing long-running multi-component parallel applications. In this paper, we have constructed a middleware framework for executing such long-running applications spanning multiple submissions to the queues on multiple batch systems. We have used our framework for execution of a foremost long-running multi-component application for climate modeling, the Community Climate System Model (CCSM). Our framework coordinates the distribution, execution, migration and restart of the components of CCSM on the multiple queues where the component jobs of the different queues can have different queue waiting and startup times.**

I. Introduction

Computational grids have been increasingly used for executing large scale parallel applications [1]–[3]. Many current large-scale grid frameworks [4], [5] consist of multiple distributed sites with each site having one or more clusters or Massively Parallel Processors (MPPs). Each cluster is typically a batch system associated with batch scheduling and queueing policies. Thus, many current grids are composed of multiple batch systems.

In this paper, we consider execution of long running multi-component applications on grids with multiple batch systems (*batch grids*, for brevity). We submit component jobs of the application to different queues of a batch grid. The jobs on different queues can have different queue waiting times and hence different start-up times. Hence the number of *active* batch systems available for execution can vary at different times. Batch queue systems are also associated with limits for execution time for a job. The execution time limits are typically few days while long-running applications can execute for several weeks. Hence a long running application should span multiple submissions in each site where the application should be checkpointed and continued from the previous submission. Hence, a framework that supports execution of multiple components of an application across a time-varying set of active queues is needed for the execution of long-running multi-component applications on batch grids.

Our work involves the development of such a middleware framework for long-running multi-component applications. We have employed a novel execution model that involves making use of all active resources by scheduling and rescheduling the application components whenever the set of active batch systems changes. Our middleware framework automatically coordinates the execution of the components on the different queues, determines the allocation of processors in the batch systems for the components, schedules and reschedules the components, checkpoints and migrates the application to a different set of available systems, and transfers checkpoint data on the new set of systems for continuation of the application. Our application execution framework thus ensures maximum utilization of resources available to the application.

II. Related Work

There are several significant grid middleware frameworks, like GrADS [6], [7], Cactus [8], [9], GridWay [10] that support checkpointing, migration

This work is supported by Ministry of Information Technology, India, project ref no. DIT/R&D/C-DAC/2(10)/2006 DT.30/04/07

The student author is Sivagama Sundari M

and sometimes reconfiguration of long-running jobs executed on dynamic resources. Some recent work [11], [12] present frameworks for execution of traditional, unmodified, legacy applications. However, none of these efforts provide a framework for execution of an application job co-allocated across multiple batch systems and dynamically reconfigured.

In our previous work, we have conducted simulation studies of the performance of the execution of a long-running application across batch systems. In this work, we present a middleware framework for such execution. To our knowledge, ours is the first work on building a framework for execution of a long running multi-component application where the different components are submitted multiple times to different batch queues. The execution of a component is stopped within the execution time limit associated with its submission to a batch queue and the component is submitted again to the same or different batch queue and continued from its previous execution.

In our model, we attempt to execute multi-component jobs using a given set of local queues of different clusters with existing and possibly different job execution policies. While submissions are made to the queues of all the clusters, an execution of the multi-component application can proceed when at least one of the clusters become available for execution.

III. Methodology

In this section, we describe the application and our grid middleware framework.

A. A long-running Multi-Component Application: CCSM

In this work, we consider a classic and foremost example of a multi-component application, Community Climate System Model (CCSM) [13], a global climate system model from National Center for Atmospheric Research (NCAR) [14]. It is a MPMD application consisting of five components, namely, atmosphere, ocean, land and ice and coupler. The first four components are climate simulation models corresponding to various climate subsystems. These components periodically exchange data, typically after every few simulated-climate-hours. The coupler is a component which coordinates the communications between the other model components. Each of the components is a moldable parallel application, i.e., it can be executed on variable number of processors.

Fig. 1. **Grid MiddleWare Framework for Long-running Multi-component Applications**

CCSM is typically executed for long periods to simulate climate systems for multiple centuries with execution times of several weeks. To support such long running simulations, CCSM contains *restart* facilities where the application can be made to store its execution state as *restart dumps* at periodic intervals or after some simulated days and simulations for an execution can be continued from the previous executions using the *restart dumps* of the previous executions.

B. Middleware Framework

Our execution framework primarily comprises of three component daemons: a *coordinator* that determines mapping of components to the batch systems, and schedules and reschedules the component executions on the systems, a *job monitor* on each front end node of batch systems that interfaces with the coordinator, and the *job submitter* on the front end node that repeatedly submits a CCSM job upon completion of the previous CCSM job. Our framework also consists of a *CCSM job script* which executes and re-executes MPI jobs on a system corresponding to specified mappings of components to processors of the different active batch systems at various points of time. The architecture is illustrated in Figure 1.

In this section we briefly describe the three components, their primary functions and their interactions.

1) **Coordinator**

The coordinator daemon is the most significant daemon and is executed on a location that is accessible from the front-end nodes of all the systems. It contains all the global information about the framework. This includes information that stays constant throughout the duration of an experiment as well as those that vary. The number and location of queues, the number and characteristics of the components, the locations of executables and restart files, etc. are some of the static information known to the coordinator. The set of active queues, the times at which the queues had become active, the previous and current CCSM configuration, etc. are some of the dynamic information contained in the coordinator.

Since the coordinator has knowledge of the state of the entire system, it can take actions and/or instruct other daemons to take actions. Some of the actions taken by the coordinator include determining the mapping of components to batch systems, scheduling and rescheduling component executions, transferring restart files, etc.

2) **Job Monitor**

The job monitor daemon monitors the local behavior of the job on each site. A job monitor daemon is started corresponding to each queue used in the framework on the front-end node of the respective system. The job monitor notifies the coordinator daemon of local events like (i) the queue becoming active; (ii) the active job on the queue is close to its execution time limit, and (iii) the current execution has been stopped and the job is ready for reconfiguration. It does these by sending *START*,*STOP* and *STOPPED* messages respectively.

The job monitor also processes the configuration data supplied by the coordinator at every reconfiguration event and ensures that the active CCSM job is suitable reconfigured.

3) **Job Submitter**

The job submitter is a daemon that runs on the front-end nodes of each batch system. One job submitter is started per batch queue. The job submitter submits the CCSM job script continually ensuring that at any time exactly one such job exists in the queue (active or inactive). The job script comprises of a loop until timeout. In the loop, the job script waits for a *component-config file* and executes using MPI, the components on the processors as specified in the component-config file.

C. Component Interactions

An application job is submitted to each of the batch systems with a request for a specific number of processors by the Job Submitter. We refer to a job submission to a queue and the corresponding batch system as becoming *active* when the job completes waiting in the queue and is ready for execution. When a job on a batch system is active, 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

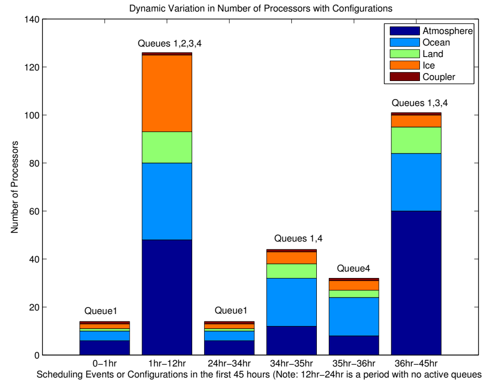

Fig. 2. **Dynamic Variation in Number of Processors**

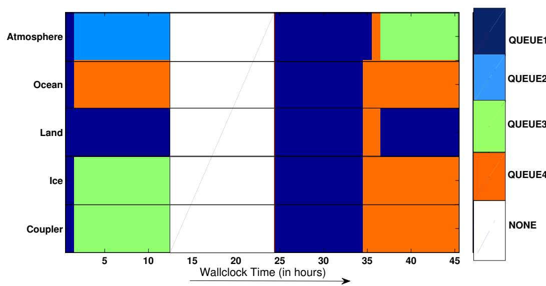

Fig. 3. **Dynamic Variation in Component to Queue Mapping**

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.

The interactions between the components of our framework are illustrated in Figure 1.

IV. Experiments and Results

We tested our 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 [15]: one queue of size 14 on fire-16, a queue of size 48 on fire-48, two queues of sizes 32 and 64 on varun.

External loads were simulated by submitting synthetic MPI jobs to the queueing systems based on the workload model developed by Lublin and Feitelson [16]. 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.

As the jobs on each of the four queues became active and inactive, the CCSM runs were automatically reconfigured and restarted by our framework. Our framework executed CCSM for a period of 3 days during which climate of 3 years, 7 months and 21 days was simulated.

The number of processors of the CCSM components on the different batch queues during the first six configurations (45 hours) of our experiment are shown in Figure 2. As can be seen the component sizes and hence the number of processes used for CCSM execution varies with the set of active queues.

Figure 3 shows the location of execution of various components along the execution time-line as the configurations change. The white region in the figure indicates a phase during the experiment during which there were zero active systems, i.e., when the CCSM jobs on all the batch systems where waiting in the queue. The number of different colors (except white) along the vertical corresponding to a time-instant indicates the number of active sites. As can be seen, the number of active sites in the figure takes all the possible values from 0 to 4.

The figures show during the first hour that queue1 is active and all the components are executed on a very small number of processors within queue1. Now, queues 2 to 4 all become active within the time it takes for the CCSM job on queue1 to stop and become ready for reconfiguration. Hence, the next configuration has components executing on all 4 queues. Atmosphere and Ocean are automatically migrated by the framework from queue1 to queue2 and queue4 respectively, while Ice and Coupler are both migrated from queue1 to queue3. Note that, at this point, land is not migrated

but is executed on a larger number of processors on queue1 itself as indicated in the Figure 2. After around 11 hours of execution on this configuration, all queues are close to time-out. All batch systems at this stage become inactive as indicated by the white region. At the 24th hour, queue1 becomes active and all components are migrated to queue1. The configuration here is similar to the first configuration as can be seen in Figure 2. In the 34th hour, when queue4 also becomes active, all components except atmosphere migrate from queue1 to queue4, and so on.

Thus, the experiment has demonstrated that our middleware framework can be effectively used for a robust long-running execution on multiple clusters with independent batch queueing systems across an open network. We have also demonstrated that it can handle large dynamic variations in the number of active queues and processors at each reconfiguration event, thus supporting effective use of the resources available on all the sites.

V. Conclusions and Future Work

In this work, we have developed a middleware framework for execution of multi-component applications on batch grids consisting of multiple batch systems. The framework was demonstrated with a foremost multi-component application, CCSM. The framework performs complicated tasks including coordination of the different startup times of the component jobs on the different batch queues and rescheduling of the component jobs based on the number of active systems for execution. We have demonstrated the framework with a four-site execution. In future, we plan to investigate different strategies for scheduling and rescheduling multi-component jobs on batch systems.

References

[1] C. Mueller, M. Dalkilic, and A. Lumsdaine, "High-Performance Direct Pairwise Comparison of Large Genomic Sequences," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 8, pp. 764–772, 2006.

[2] X. Espinal, D. Barberis, K. Bos, S. Campana, L. Goossens, J. Kennedy, G. Negri, S. Padhi, L. Perini, G. Poulard, D. Rebatto, S. Resconi, A. de Salvo, and R. Walker, "Large-Scale ATLAS Simulated Production on EGEE," in *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, 2007, pp. 3–10.

[3] M. Gardner, W. chun Feng, J. Archuleta, H. Lin, and X. Mal, "Parallel Genomic Sequence-Searching on an Ad-hoc Grid: Experiences, Lessons Learned, and Implications," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006, p. 104.

[4] "TeraGrid," http://www.teragrid.org.

[5] "UK e-Science," http://www.rcuk.ac.uk/escience/default.htm.

[6] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-crummey, D. Reed, L. Torczon, and R. Wolski, "The grads project: Software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, pp. 327–344, 2001.

[7] S. S. Vadhiyar and J. J. Dongarra, "Self adaptivity in grid computing," *Concurrency & Computation: Practice & Experience*, vol. 2005, 2005.

[8] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf, "The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment," *International Journal of High Performance Computing Applications*, vol. 15, p. 2001, 2001.

[9] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H. christian Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf, "Cactus tools for grid applications," 2001.

[10] E. Huedo, R. S. Montero, and I. M. Llorente, "A framework for adaptive execution in grids," *Softw. Pract. Exper.*, vol. 34, no. 7, pp. 631–651, 2004.

[11] L. Du, Y. Wu, and C. Wang, "Component based legacy program executing over grid," in *GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 558–565.

[12] N. Markatchev, C. Kiddle, and R. Simmonds, "A framework for executing long running jobs in grid environments," in *HPCS '08: Proceedings of the 2008 22nd International Symposium on High Performance Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 69–75.

[13] "Community Climate System Model (CCSM)," http://www.ccsm.ucar.edu.

[14] "The National Center for Atmospheric Research (NCAR)," http://www.ncar.ucar.edu.

[15] "Pbs gridworks: Openpbs," http://www.openpbs.org.

[16] U. Lublin and D. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, 2003.