Executing Long-running Multi-component Applications on Batch Grids

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

***Abstract*— Computational grids are increasingly being used for executing large multi-component scientific applications. The most widely reported advantages of application execution on grids are the performance benefits, in terms of speeds, problem sizes or quality of solutions, due to increased number of processors. We explore the possibility of improved performance on grids without increasing the application's processor space. For this, we consider grids with multiple batch systems. We explore the challenges involved in and the advantages of executing long-running multi-component applications on multiple batch sites with a popular multi-component climate simulation application, CCSM, as the motivation. We have performed extensive simulation studies to estimate the single and multi-site execution rates of the applications for different system characteristics. Our experiments show that in many cases, multiple batch executions can have better execution rates than a single site execution.**

I. Introduction

Large scale scientific applications are increasingly being executed on computational grids. For these applications, grids have been used to provide feasibility to solve much larger problem sizes than those that can be solved on a single site [3], [5], [6], [11], to explore larger search space of different parameters of a problem [1], [8], to improve the quality of the solutions [9], [12] and to enhance application performance [3], [5], [11]. Most of these grid benefits are primarily due to increase in the number of processors available for execution. However, many scientific applications have limitations in scalability to large to very large number of processors. In this work, we focus on yet another potential use of grids where the processor space for application execution is not increased. Specifically, we focus on grids with multiple batch systems (we refer to these as batch grids) and show that employing multiple batch systems can improve the execution rate of long running multi-component applications.

CCSM (Community Climate System Model) [4] is one of the most prominent examples of long-running multi-component scientific applications. It has five components: the climate simulation models for atmosphere, land, ocean and sea-ice and a coupler that coordinates the information flow among the models. The simulations are typically performed for several centuries of simulated-time and can take several weeks or months to execute. While the components are moldable with respect to the number of processors, they are not very scalable and some of them cannot be executed beyond certain number of processors determined by their climate data-resolution.

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

Sivagama Sundari M is the student author

II. Application Execution on Multiple Batch Systems

Parallel batch systems provide space-sharing of available processors among multiple parallel applications or jobs. These batch systems employ queues in which the incoming parallel applications are queued before allocation by a batch scheduler to a set of processors for execution. Hence, the overall response time of an application is the sum of its queue waiting time and execution time. In order to discourage some jobs from engaging resources for extremely long durations, most batch systems impose a maximum execution time limit per submission. Therefore, very long running applications like CCSM require multiple submissions and incur queue waiting times for each submission. While requesting larger number of processors could speed up the execution of many scalable applications, larger requests generally incur longer queue waiting times. This is because smaller requests can be backfilled more easily. Even in the First-Come-First-Serve (FCFS) policy where backfilling is not allowed, a job with smaller request will be allocated sooner than a job with large request on completion of jobs with smaller processor requirements. Figure 1 shows the average queue wait times for jobs with different processor requirements on an IBM SP2 system in SDSC (San Diego Supercomputer Center), the job traces for which were obtained from the logs maintained by Feitelson [7]. From the figure, we find that in general the queue waiting times of the jobs increase with their request sizes. In our work, therefore, we consider splitting an application request for a large number of processors, into smaller requests and submitting these smaller multiple requests to different batch systems possibly located at different sites. Some of the factors that affect the feasibility and the possibility of better performance with multiple site execution are mentioned below.

1) **Application Factors** While some parallel applications like the NAS Parallel Benchmarks (NPB) [2] with absolute processor requirements cannot be decomposed into sub applications with arbitrary processor requirements, others like Parameter Sweep Applications can be. Yet other applications like the multi-component CCSM by design have a fixed number of well-defined components. Further, decomposing communication-intensive applications into smaller parallel applications and submitting to different batch systems will result in frequent and heavy communications between the batch systems. Since these batch systems can be located in different sites of a grid, the heavy communication between the sub applications will lead to much larger execution times of the applications than when executed on a single system. Thus the overall response time of the application can significantly increase in spite of the reduction in queue waiting times. Multi-component MPMD applications like CCSM consist of components which are parallel applications themselves. In these applications, the components are loosely coupled and communications between components are lighter and less periodic than within components. Hence, they have a potential for multiple-site executions.
2) **System Factors** Processors in a single batch system are connected by high speed links while processors of different batch systems are typically connected by WAN links and other high latency networks. The greater the speed of the interconnects, the lower the inter-site execution time and the better the response times of multiple-site executions of communicating sub-applications. As mentioned earlier, the advantage of multiple-site executions that we are interested in is mainly due to the in-

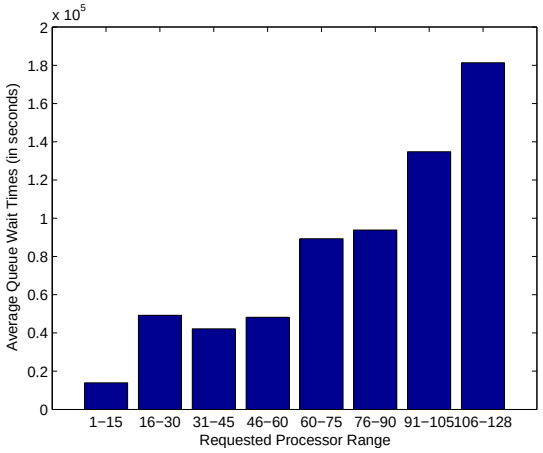

Fig. 1. Average queue wait times for jobs on a SDSC system

crease in queue waiting times with the request size. The queue waiting times are also affected by several other factors like the workload characteristics at each site, the queueing policy adopted at each site, the maximum execution time limit imposed by each site, etc.

III. Multiple-Batch Execution Policies for Multi Component Applications

Due to its long-running nature, the Multi-Component Application (MCA) execution completes over several submissions in a batch system. Since the results of the last submission would be needed to continue execution in the next submission, the submissions are sequentialized. Hence, at any point of time there is only a single submission of MCA and it is either being executed or is queued. Note that the execution times of each of the previous submissions would be equal to the maximum execution time allowed in the batch system.

One of the most significant challenges in multi-component executions on multiple batch systems is that queues become available at different times and components are generally required to run simultaneously. While some multiple-site scheduling systems support coallocating such jobs, most sites do not. Hence, we consider different possibilities or policies of execution. For simplicity, we restrict ourselves to a single job request (of MCA) in each system at any point of time. In the description that follows, we use the term “active” to indicate the state of a batch system in which our job has has been assigned the requested number of processors and can immediately start executing. We use the term “inactive” to indicate the state when our job is waiting in the queue for resources.

Following are the different execution policies we consider.

1) **WaitForAll (WFA)** We assign one component to each batch system and execute the MCA only when all the systems are active. The active systems idle at other times waiting for the inactive systems to become active. While this strategy is simple, it could keep the system resources engaged for the entire duration of a submission without performing any useful work.
2) **WaitTillThresholdandAbort(WTTA)** This is a slight modification of the previous strategy to prevent long continuous idling. Again we execute the MCA only when all queues are active. However, any partial set of active queues wait for all queues to be active only upto a threshold time-limit.
3) **WaitTillThresholdandExecute(WTTE)** In the previous strategy, a large threshold would mean resources uselessly engaged for long durations. However, a small threshold may not give sufficient time for all queues to become active and the MCA submissions could get canceled and resubmitted very often without being executed. In order to guarantee the MCA execution, the active systems wait like in the previous strategy for all systems to become active until the threshold. When the threshold is crossed, however, instead of canceling the submissions, we begin execution of MCA on the active systems. The execution would continue until the maximum execution time limit of one of the active queues is crossed. If in the meantime, i.e. during the MCA execution, some of the inactive queues become active they are canceled, the resources returned and new submissions are made on those systems.
4) **NoWait** Finally, we consider the strategy of execution without any idling or surrender of active resources. In this strategy, we use all available resources at all times by reconfiguring the MCA whenever there is a change in the set of active systems.

IV. Experiments and Results

We have performed several simulation experiments to quantify and understand the effects of most of the factors mentioned in the previous section. Our simulation setup comprises of three major components: (i) a workload simulator, (ii) a multiple batch system simulator and (iii) the execution rate calculator as shown in Figure 2.

The workload simulator produces a list of jobs with submission time, processor request size and expected execution time for each job. Although such lists could be obtained from real logs of various supercomputing sites, using a workload model

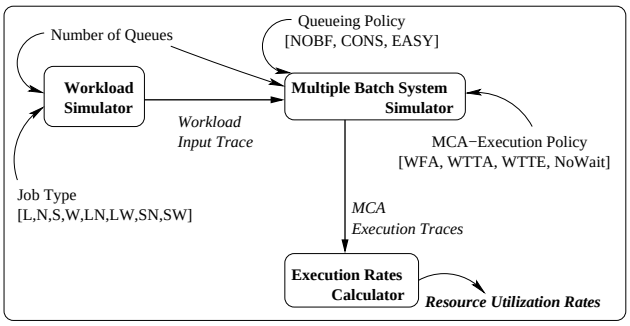

Fig. 2. Simulation Setup

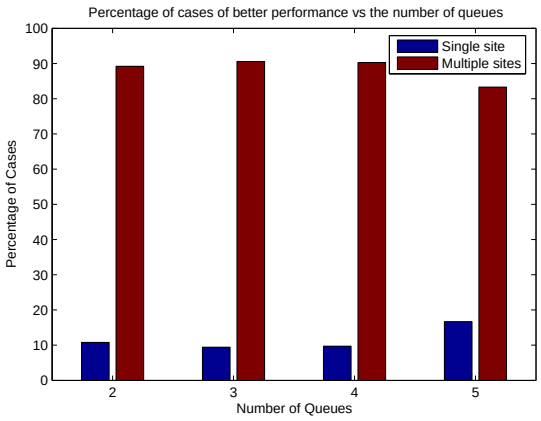

Fig. 3. Percentage of cases in which each mode outperforms the other

to generate the job sequence enables us to vary workload characteristics. We can then include their effect on the performance of a parallel application when executed across multiple batch queues. We used the workload model developed by Lublin and Feitelson [10]. Job processor requirements, runtimes and arrivals are modeled based on a two-stage uniform distribution, a hyper-Gamma distribution and a Gamma distribution, respectively. The model parameters are preset to values representative of real logs of supercomputers. For all our job traces, we specified the maximum processor requirement of the jobs as 128 processors and maximum execution time of 2 days. We categorize a job as long (L) or short (S) based on its execution time and as narrow (N) or wide (W) based on its processor equirements. By tuning parameters that decide the ratios of number of L to S jobs and the number of W to narrow N jobs, we generated 8 different categories of job traces: L, S, W, N, SN, SW, LN and LW.

Our batch system simulator is an MPI program with number of processes equal to number of batch systems. Each process simulates the queueing policy of the system with a workload trace obtained from the workload simulator. The MCA job request is included along with the workload using one of the four MCA-execution policies described in the previous section. Three different queueing policies, namely, FCFS, CONS (conservative backfilling) and EASY (EASY backfilling) were considered: The execution rate calculator uses the execution trace output of the simulator to calculate the resource utilization rate (RUR). RUR is the number of processor-hours available for execution of the MCA per day. For a scalable application RUR is directly proportional to the application execution rate.

An MCA with 5 components each of size 25 was considered for our experiments. For a given experiment with a set of batch systems, the components of MCA were distributed in all possible ways to all the batch systems. Different experiments were performed where the number of batch systems were varied from 2 to 5. Corresponding to each multiple batch experiment, a single batch system experiment was performed. The queue characteristics including the scheduling policy for each system were randomly chosen with uniform distribution. A total over 6000 experiments were performed.

The multiple queues had better RUR values than the single queue only with the WTTE and NoWait policies. And as expected the NoWait policy performed the best.

Figure 3 shows the percentage of cases in which the multiple site had a higher RUR than the single site execution and vice versa. The results are shown for different number of queues with the NoWait policy followed for multiple site executions.

Figure 4 shows the average fraction of the total simulation time of each queue spent in executing the MCA on various number of active queues. Note that the fraction of time in which all queues are active is lower than the fraction of time in which some queues are executing. Hence, the WFA and WTTA policies, that require all systems to be active for execution, gave poor performance.

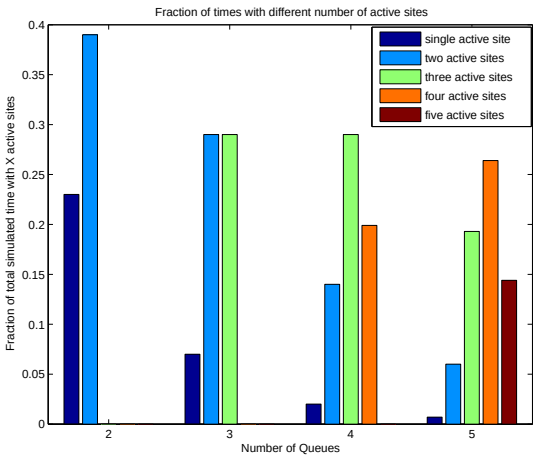

Fig. 4. Durations of different number of active sites involved in MCA execution

V. Future Work: Practical Implementation

We are planning to implement the best performing NoWait strategy with CCSM on real batch systems. We are currently developing more refined models for inter-site and intra-site CCSM executions to perform more CCSM-specific simulations to identify application-specific advantages and bottlenecks.

In our work, we do not expect any special middleware or any special considerations from the batch schedulers, and focus on execution of the MCA (CCSM) in the existing environment. However, we do assume that components executed on different batch systems can communicate witheach other. This assumption is reasonable since few MPI (Message Passing Interface) communication libraries including PACX-MPI and MPICH-GX support communications between MPI processes started on different batch systems by means of special communication processes or proxies executed on the front-end nodes of the batch systems. We are currently performing experiments with CCSM across multiple disparate sites using the PACX-MPI framework.

While a practical implementation of the multiple-batch site is very much feasible, several implementation overheads such as those associated with execution time-outs, notifications, scheduling, initialization, restart-dumps and restart-transfers will have to be minimized for the solution to be viable.

References

[1] C. An, M. Taufer, A. Kerstens, and C. Brooks III. Predictor@Home: A P̈rotein Structure Prediction Supercomputer' Based on Global Computing. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):786–796, 2006.

[2] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, Nasa Ames Research Center, December 1995.

[3] W. Chrabakh and R. Wolski. GridSAT: A Chaff-based Distributed SAT Solver for the Grid. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 37, 2003.

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

[5] S. Dong, N. Karonis, and G. Karniadakis. Grid Solutions for Biological and Physical Cross-Site Simulations on the Teragrid. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[6] 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*, pages 3–10, 2007.

[7] Logs of Real Parallel Workloads from Production Systems. `http://http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`.

[8] 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*, page 104, 2006.

[9] L. Han, A. Asenov, D. Berry, C. Millar, G. Roy, S. Roy, R. Sinnott, and G. Stewart. Towards a Grid-Enabled Simulation Framework for Nano-CMOS Electronics. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 305–311, 2007.

[10] 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.

[11] C. Mueller, M. Dalkilic, and A. Lumsdaine. High-Performance Direct Pairwise Comparison of Large Genomic Sequences. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):764–772, 2006.

[12] C. Stewart, R. Keller, R. Repasky, M. Hess, D. Hart, M. Muller, R. Sheppard, U. Wossner, M. Aumuller, H. Li, D. Berry, and J. Colbourne. A Global Grid for Analysis of Arthropod Evolution. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 328–337, 2004.