

Adapting an Application for Use in a Condor Based Parameter Sweep on TeraGrid

P. A. Cheeseman, M. W. Deem, D. J. Earl, and William I. Whitson

Abstract— This paper describes the method by which a single case application was adapted to usage as the core of a high throughput parameter sweep. Performance characteristics of the application, which were considered in planning the adaptation, are first described. The steps towards adapting the code to high throughput batch processing systems are then discussed with an emphasis on adaptation to Condor[1] as deployed in the Purdue TeraGrid resources. A summary of the change in performance characteristics of the application is presented followed by concluding remarks.

Index Terms— ASTA, Condor, Parameter Sweep

1 INTRODUCTION

Frequently in computational pursuits, programs are written to assist in the solution of one, or a small set of related problems. Once developed, however, a program may be seen to be adaptable to other purposes.

Such evolutionary adaptations often lead to more frequent use of a program's descendants or to descendants which require significantly more resources during their execution. This is the case for a component of the *Zefsa II* program set developed by M. Falcioni and M. W. Deem[2].

Briefly, a part of *Zefsa II*, referred to generically in this article as *banneal*, was seen by Profs. Deem and Earl to be useful as a tool for identifying hypothetical zeolite structures. The scientific goal of the project is to search through crystallographic space for new potential zeolite structures. Zeolites are porous crystalline materials that have a wide range of uses in, for example, catalysis, ion exchange, and molecular sieving applications. The procedure for identifying hypothetical zeolite structures, described below, may be referred to as a parameter sweep in simplistic terms.

The scope of the parameter sweep, proposed to TeraGrid by Profs. Deem and Earl, was known to require more than 200,000,000 executions of *banneal* in its distributed state. The anticipated and observed impact of this computation led to eventual modification of *banneal* in order to facilitate more efficient use of the program within Condor, or for that matter, any batch processing environment.

The changes made to *banneal* are general enough in nature to warrant their discussion as the subject of this article. They differ from the usual changes one might make to a scientific code in that their purpose is almost entirely related to easing the burdens associated with large scale use of the application.

2 CHARACTERISTICS OF THE COMPUTATION

Banneal is used to perform a simulated annealing process using a Monte Carlo algorithm. Initially, the operational procedure for performing the computations consisted of executing *banneal* a number of times, specifically 100, for each parameter set. Each of the 100 executions, for each parameter set, was performed with a different seed number. For each seed value, or *cycle*, a result file requiring about 20 Kbytes was produced.

Parameter sets were organized in groups identified with the crystallographic *space group* being considered. The number of sets for a given group varied from roughly 7,000 to 30,000. Data files associated with each set resided in a set specific directory beneath a single parent. Groups were packaged as compressed tar files.

Early progress on the project was achieved by submitting jobs to the Condor system which executed *banneal* for each individual seed value within a single Condor job. Repeated executions were achieved by means of an *execution script* which, in addition to executing the application, provided a degree of self-checkpoint capability. This approach immediately avoided the overhead of sending a job to the Condor system for each individual seed.

With approximately 10,000 sets (1M cycles) complete, the following characteristics of the computation were noted.

- Execution times per set were not well defined. While many 100 cycle jobs could be expected to finish in an hour or less, others could continue for many hours. Some jobs were manually terminated after twelve or more hours. The distribution of execution times for the initial group processed at RCAC is shown in Figure 1 with a summary breakdown in Figure 2.
- Average execution times per group could also vary significantly but appeared to be bounded by a maximum of approximately three hours.
- Computational efficiency of the program varied depending on what compiler was used to build the executable. For this application, the Intel Version 8 and 9 C compilers were observed to produce the fastest executable for Intel based Linux systems. The performance improvement was more than 30% when compared to various versions of gcc. For this reason, the Intel compiler suite was requested by Profs. Deem and Earl in

• Michael W. Deem Department of Bioengineering,, Rice University, 6100 Main Street – MS 142, Houston Texas, 77005-1892
 • David J. Earl, Department of Chemistry, University of Pittsburgh,322 EberlyHall,219 Parkman Avenue, Pittsburgh, Pennsylvania, 15260
 • P. A. Cheeseman, Rosen Center for Advanced Computing, Purdue University, 302 Wood Street, West Lafayette, Indiana, 47906-3560.
 • William I. Whitson, Rosen Center for Advanced Computing, Purdue University, 302 Wood Street, West Lafayette, Indiana, 47906-3560.

their proposal.

- Result handling could become a major concern. For 7,000 parameter sets, 700,000 result files would be produced requiring about 3 GB if saved with compression.
- Exceptional exits, such as errors in data, were virtually non-existent. A complete result file could be expected from each cycle. If a result file was not present for a cycle, that fact was indicative of the need to re-execute the cycle in virtually all cases.

Unadapted Execution Times

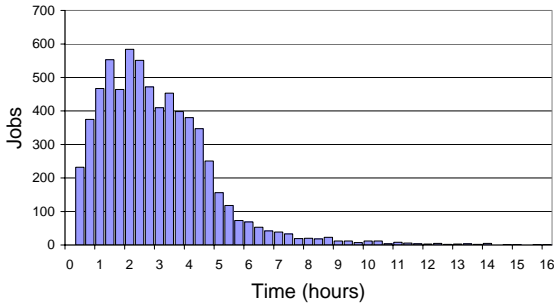


Fig.1. Distribution of Execution Times for Control before Adaptation.

Unadapted Execution Time Breakdown

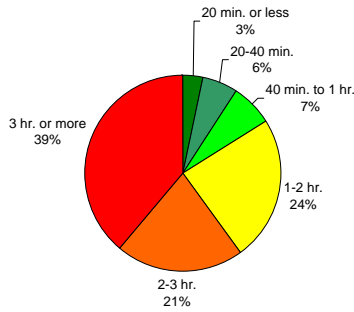


Fig.2. Breakdown of Execution Times for Control before Adaptation.

3 CHARACTERISTICS OF CONDOR UNIVERSES

The computation described here was expected to use one of two particular Condor *universes*[3], either the *vanilla* or *standard*. Within either universe, certain features and restrictions would have direct impact on decisions made regarding the methods used to sustain production computation.

Under either universe, jobs would be subject to the possibility of preemption by non-Condor processes on the execution platforms. The presence of a preemption capability, however, gives rise to another fundamental feature of Condor. Jobs running under Condor are not subject to resource limits in the usual sense. If a job requires, for instance, a wall time limit, that limit must be self imposed.

The features of each universe pertinent to this article are described below.

3.1 The Vanilla Universe

When executing under the vanilla universe, jobs may run much as they would under other batch processing systems. The following features of the vanilla universe had a direct bearing on the adaptation of the application.

- Preemption of a vanilla universe job is achieved by either suspension or *eviction*. Eviction refers to stopping the job and restarting it from the beginning. Unless a job is self checkpointing, all work is lost. Eviction can be, therefore, very wasteful. Jobs which cannot restart near the stage at which they were evicted have simply left the unused resources scavenged by Condor, again, unused.
- Jobs running under the vanilla universe may spawn child processes. In other words, the executable for a vanilla universe job can be a shell script.
- The application need not be linked with the Condor run-time library.

A good deal of preliminary work for this application was performed under the vanilla universe. During that period, eviction rates of more than 50% for the jobs submitted were commonly observed as indicated by Figures 3 and 4.

Unadapted Eviction Rates

Data for 170 Batches (326,041 Jobs)

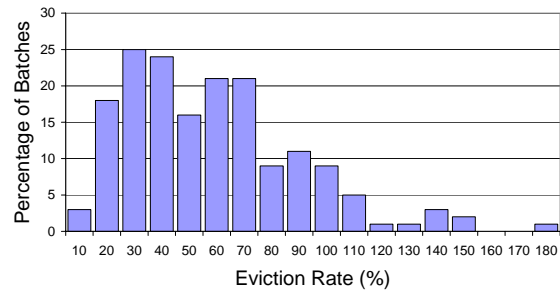


Fig. 3. Distribution of Eviction rates Before Program Adaptation

Unadapted Eviction Rate Breakdown

Data for 326,041 Jobs

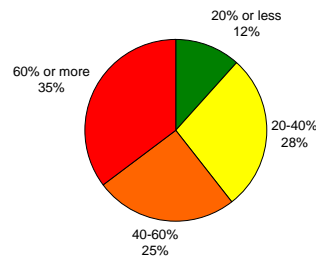


Fig. 4. Breakdown of Eviction rates Before Program Adaptation

3.2 The Standard Universe

The standard universe is the more attractive of the two job execution environments discussed here when considering the case of long running jobs. Features of the universe which mattered most for this project included the following.

- Preemption by checkpointing is possible for standard universe jobs in addition to eviction and suspension.

- The executable application for a standard universe job cannot spawn sub-processes. Scripts are therefore not candidates for the standard universe.
- The executable for standard universe jobs must be linked using the Condor run-time libraries. Linking the application with the Condor run-time, was not supported for applications compiled with the Intel compilers.

4 THE PURDUE CONDOR ENVIRONMENT

The TeraGrid Condor facility at Purdue continues to present the user with a growing, amorphous mixture of platforms. Unlike some other Condor installations, the Purdue installation matches job requirements with platforms that are *not* dedicated entirely to serving Condor.

Some additional notes about the Purdue Condor pools must be mentioned before describing the alternatives considered towards improving throughput and performance for this project.

- The Linux platforms made available via Condor may vary significantly in terms of hardware. For instance, CPU clock speeds might vary from 2 to 3 GHz.
- Many of the platforms available to Condor users are within batch processing clusters scheduled via PBS[4]. The eviction rates observed during the early phases of this project were directly related to the fact that Condor jobs were being preempted by PBS submissions. In other words, jobs running within the Purdue Condor facility for long periods of time are almost certainly destined for checkpoint or eviction. *The important point to remember here is that evictions occur due to circumstances beyond the user's control.* The one thing under the user's control which does correlate with eviction probability is job impact, or in the case of banneal, the job execution duration. The less time a job is in execution, the less likely it is to be evicted.
- Early experience with the Condor facility indicated that the number of jobs executing in the system would vary between several hundred to more than 1000. This fact gave rise to concerns about the load that would be shouldered by the submission host.

5 ADAPTATION ISSUES

Given the nature of the application, Condor, and the characteristics of the Purdue Condor installation, the following issues were regarded to require attention.

- Parameter sets requiring a large amount of computation time, could conceivably never finish within the vanilla universe due to repeated eviction. If the vanilla universe became the production choice, a self-checkpointing capability would be required as part of a mechanism to insure completion of long computations.
- Parameter sets using excessive amounts of compute time needed to be detected and terminated automatically, rather than manually (by watching them individually). Tracking down runaway jobs when there are, say, 1500 jobs in the queue is a tedious undertaking

to say the least.

- To use the standard universe efficiently, the code would require modification to process multiple cycles during execution. In its unmodified state, the application would require submissions of a single job per cycle. The number of jobs submitted for the project then increases by a factor of 100. Such conditions would be intolerable in the case of parameter sets for which a single cycle executed for a few seconds since the overhead of job handling and submission would outweigh the actual computation impact.
- In either universe, some attention was needed towards making sure that the amount of *computation* completed for a given cycle would be roughly the same regardless of the CPU speed of the execution platform. If the application were to monitor execution time, it would also need to extend its self imposed limits when running on slower processors.

Summarizing the above issues reduces to a simply stated solution. There was a need to alter the application so that it would run long enough *per job* to make submission worth the overhead while, at the same time adding controls to prevent excessively long executions.

In the vanilla universe, the code would need work to limit its exposure to eviction. Eviction could be reduced, of course, by tuning job duration to a reasonably short time period characteristic of the Condor pool.

6 PROGRAM MODIFICATIONS

First of all, early stage modifications to the execution script were made to automatically discontinue simulations that ran excessively long. With this change in job management, missing result files were no longer conclusive evidence of a failure condition. To confirm premature self-termination of the computation, an informational file was created by the execution script indicating premature self-termination.

The script based controls, however, were less than satisfactory stop-gap measures. They were installed to allow computation to continue while better alternatives for improving job throughput and efficiency were investigated. These controls, furthermore, perpetuated the requirement that the application be executed in the vanilla universe and created overhead on the execution platform that could rival the cost of cycle execution.

The following modifications to the core application were, therefore, proposed. Their primary purpose was to eliminate the need for the execution script.

- Modify the program to cycle through all seed values for a parameter set.
- Incorporate the capability to continue partially completed sets. Eviction and other premature stops in set processing should be weathered more gracefully.
- Add the ability to process multiple or partial parameter sets. Job duration could then be adjusted to longer periods in cases where typical sets complete in minutes or shorter periods when sets required more than, say, an hour. In other words, make average job duration reasonably tunable to the execution (eviction) environment.
- Create a self control mechanism by which the program

could limit execution duration. The proposed change provided for limits based on execution times per cycle and net execution time. The code would shut down at the end of a cycle if a limit condition was determined and leave an indication of the reason in the result file.

- Deal with contingent code changes necessitated by those listed above. For instance, unintended process growth (stack leaks), due to incorporating the seed cycle, would require elimination.

It must be noted here that the above changes would have made sense in any high throughput batch environment. A code which anticipates preemption, and shuts down or recovers predictably, requires less extraordinary attention when used. This is the case whether or not a thousand or so copies are running simultaneously.

Following review by all concerned parties, program changes were made and tested as production computation for the project continued.

The continuation mechanism built into the program was capable only of redoing the computation for cycles which were not completed. While this change lessened the impact of eviction on the computation, eviction was still an event to be avoided.

The controls to limit execution duration were implemented as capabilities to stop execution based on three criteria. Those criteria were the job total execution time, the time per parameter set, and the moving average of cycle times for the parameter set. A moving average of cycle times was chosen rather than a simple limit per cycle due to the variation observed in cycle times. Since the period of the average was implemented as a program option, a desired limit per cycle could be achieved by setting the period for the moving average to one.

Experience gained during testing led to a final choice of using the vanilla universe for production computation. The inability to link Intel V9 compilations with the Condor runtime simply ruled out using the standard universe with the anticipated 30%+ decrease in performance.

Figures 5 and 6 illustrate the net impact on job execution statistics facilitated by the changes to the core application. These are data for a small control case consisting of 6707 sets. The case was processed at the beginning of the project and, again, after the adaptation was complete. Recalculation was completed by sending five jobs per parameter set instead of one. The effect on execution time was as expected.

7 LOGISTICAL ISSUES

When the code changes came into use in production runs, significant work was then performed to tune the job handling procedures associated with job submission, result verification, and disposal of results. These *steward procedures* were created to perform the following basic steps.

- The complete tar file for a group was broken down into batches small enough to allow processing of the batch within disk quota limitations. A batch size of 2000 parameter sets was chosen, after some experimentation, so that results from a batch could be archived to CD if desired.
- Each batch was then processed by a procedure devel-

oped to submit jobs based on queue levels for the user. The procedure facilitated job submission, result validation, and packaging of results for each set. Queue levels were held to a selectable maximum for total jobs with submission being deferred if the number of jobs executing fell below 90% of the total in the queue.

- Once processed, a batch was scanned for problem conditions and reprocessed if necessary. Complete batches were repackaged in a tar file similar to that from which the input sets were extracted.

Adapted Execution Times

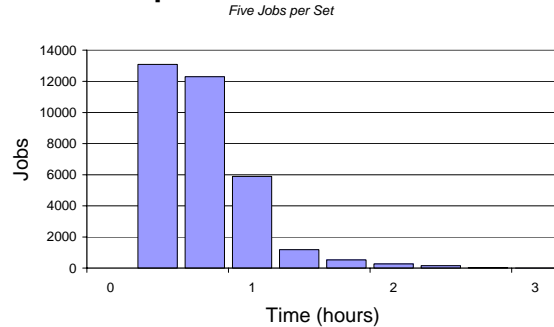


Fig.5. Distribution of Execution Times for Control after Adaptation.

Adapted Execution Time Breakdown

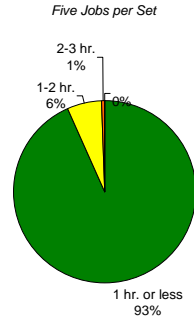


Fig.6. Breakdown of Execution Times for Control after Adaptation.

The steward procedures were designed to provide flexibility with regard to storage issues. A particular feature of their design was the ability to extract from an input set distribution resident on high capacity archival storage to batches resident on higher performance storage.

A less anticipated necessity in the evolution of the stewards was fault tolerance. Put simply, when sending jobs to approximately 2,000 loosely coupled platforms on a production basis, the probability that some of those platforms may fail operationally becomes high enough to require automatic recovery methods. As an example, this project employed NFS mounted storage for all its storage needs. The stewards, therefore, contained procedures at multiple levels to recover from NFS related failures such as automounter problems.

After the basic code modifications, the stewards were modified to process partial parameter sets per job if desired. This final step allowed regulation of the job duration in execution or, in other words, exposure to eviction. Dur-

ing the same period, a few controls were added to the code to ease the process of verifying job results, using scripts, regardless of the execution environment.

Figure 7 compares eviction rate distributions for bulk data taken before and after program modifications. While some sets displayed eviction rates much above 20%, the performance of the program was clearly better following the code changes. Batch eviction rates above 100% were eliminated completely.

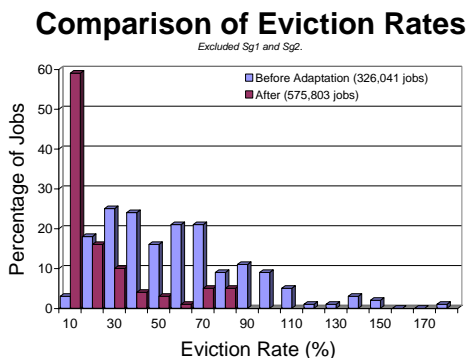


Fig.7. Comparison of Eviction Rate Distributions.

Figure 8, provided for comparison with Figure 4, displays the breakdown of post-adaptation eviction rates. Eviction rates in the range above 40%, as can be seen, decreased from 60% of the batches to 14%. Eviction rates may decrease more as the work continues and we are able to refine the job control strategies to better control job duration.

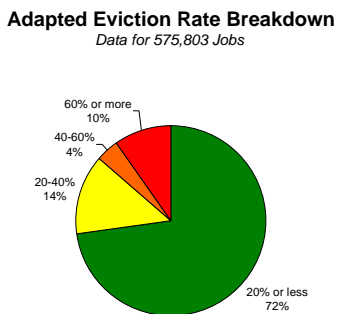


Fig.8. Breakdown of Eviction Rates after Adaptation.

8 RESULTS

At the time of this writing, approximately 1.7 million parameter sets have been processed using the Purdue TeraGrid Condor facility.

Structures resolved from the work have contributed to the searchable database[5] maintained on the web in collaboration with Professor Michael Treacy at Arizona State University. The database is approximately 65% complete.

A complete description of the project has been published in the Journal of Industrial and Engineering Chemical Research[6].

9 CONCLUDING REMARKS

During slightly less than twelve months, the Purdue Condor facility has delivered 2.3 million SUs to this project. In other words, for every hour since the project began, more than 250 hours of processor time were provided to the computation. Peak rates were much higher. This productivity is due, in large part, to looking under the hood of the core application and retooling it so job impact could be customized to match the systems available via Condor scavenging.

Every processor used in the computations would have otherwise been idle, doing nothing but drawing power while waiting to become part of an active PBS reservation or some other type of batch scheduling. All processors were scavenged by Condor rather than scheduled as they would be in a traditional batch system.

While adapting the application to efficient use of Condor has required significant effort, the fact that so much processor time was expected to be, and has been, utilized justifies that effort. The performance also speaks well of the Condor system.

The time invested in the adaptation and facilitation of production has, indeed, exceeded the 25% FTE described in the Advanced Support for TeraGrid Applications (ASTA) Overview[7]. The investment, however, has proven to be beneficial to the Resource Provider in the fact that Dr. Cheeseman was ‘on the ground’ with the project and able to deal with problems expeditiously that could have grown to system level issues. The benefit to the researchers was, at minimum, a significant improvement in throughput.

ACKNOWLEDGMENTS

This work was supported by TeraGrid allocations TG-MCA05S015 and TG-MCA05T015.

To the professional staff at RCAC who have facilitated the progress of this project, thank you for your dedication to providing reliable platforms across which these millions of Condor jobs have been processed. Finally, thanks to Preston Smith for his attention to Condor facilities at Purdue. Its 24/7 performance, under frequently difficult loading conditions, has made this computation much more a journey than an adventure.

REFERENCES

- [1] Condor Manuals,
<http://www.cs.wisc.edu/condor/manual/>
- [2] M. Falcioni and M. W. Deem. *J. Chem. Phys.* 110, 1754-1766 (1999).
- [3] Condor Manuals, Road Map for Running Jobs,
http://www.cs.wisc.edu/condor/manual/v6.8.4/2_4Road_map_Running.html#SECTION00341000000000000000
- [4] PBSPro Version 5.4.2,
<http://www.rcac.purdue.edu/rcac/aboutus/resources/learn.cfm>.
- [5] M. W. Deem, D. J. Earl, M. Treacy, "DEEM DB",
<http://www.hypotheticalzeolites.net/DATABASE/DEEM/>
- [6] D. J. Earl and M. W. Deem, "Toward a Database of Hypothetical Zeolite Structures", Eduardo Glandt special issue, invited, *Ind. & Eng. Chem. Res.* **45** (2006) 5449-5454.
- [7] Advanced Support for TeraGrid Applications,
<http://www.teragrid.org/programs/asta/>