On-demand High Performance Computing: Image Guided Neuro-Surgery Feasibility Study

Tharaka Devadithya
Indiana University
Kim Baldridge
SDSC, UC San Diego
University of Zurich

Adam Birnbaum
Amitava Majumdar
Dong Ju Choi
SDSC, UC San Diego

Rich Wolski
UC Santa Barbara
Simon Warfield
Neculai Archip
BWH, Harvard

Abstract

The emerging cyberinfrastructure holds the promise of providing on-demand access to high performance network, compute and data resources. Image guided neurosurgery is one of many applications that requires such on-demand access to resources. In this paper we have studied the feasibility of accessing such resources on-demand. An experiment was designed and carried out across five TeraGrid clusters for this study. This paper provides an analysis of the results and draws some conclusion regarding feasibility of on-demand access to high performance resources.

1. Introduction

The goal in image guided neuro surgery (IGNS) is to provide 3D image of the brain that clearly delineate anatomical structures and tumor tissue [1]. Key surgical challenges for neuro surgeons during tumor resection are to (1) remove as much tumor tissue as possible, (2) minimize the removal of healthy tissue, (3) avoid the disruption of critical anatomical structures, and (4) know when to stop the resection process. These challenges are compounded by the intra-operative shape deformation of the brain that happens as a result of tissue resection and retraction, injection of anesthetic agents, and loss of cerebrospinal fluid. The result is that the accuracy of the pre-operative plan diminishes steadily during the procedure. It is therefore of great importance to be able to quantify and correct for these deformations while a surgery is in progress, by dynamically updating the pre-operative images in a way that allows surgeons to react to changing conditions.

The procedure during the IGNS would be as follows. Images of the patient's brain are captured pre-operatively and are registered against a brain anatomy atlas. During surgery, using an intra-operative scanner, new images of the patients brain are acquired [2] and a finite element bio-mechanical model is solved [3] to construct a new visualization merging pre- and intra-operative data [4].

The dynamic updating of these images needs to be performed in a short period. The response time is critical, since the surgeons could be waiting for the next visualization model in order to proceed with the surgery. It is very likely that in a hospital surgery room, there would be limited compute power, which would result in a less accurate model. In order to obtain a model with improved accuracy and precision, the application needs to be run on a multi-teraflop machine. However, having such machines at each and every hospital is not feasible.

The solution is to remotely access large-scale computational resources, which are generally available at the major supercomputer centers, in the most efficient and reliable manner. In this paper, we examine the feasibility of using remote supercomputers to achieve time-critical neurosurgery procedures.

2. Experiment

2.1. Scope

The response time for executing a remote computational job for neurosurgery depends on the time required to send the intra-operative image data to the supercomputer, solve the finite element model, and send the results back to the hospital. As a first phase of this experiment, we are limiting our scope to the computation part corresponding to solving the finite element model.

However, the nature of the surgery demands that the entire process, including the communication, be performed within ten minutes. Some preliminary experiments showed us that the time to both receive the intra-operative images, from Brigham and Women's Hospital (BWH) at SDSC, and send visualization model back, from SDSC to BWH, take less than four minutes. Therefore, we targeted six minutes to acquire resources on a parallel cluster and complete simulation.

2.2. Objective

The algorithm for solving the finite element model is assumed to be scalable, such that better quality results are obtained with higher numbers of CPUs. Therefore, while the primary objective of this experiment was to have job completion within the given time frame, another major objective was to determine the maximum number of CPUs that can be allocated during that time. Our experiment targeted determination of the best possible job outcome, i.e., the job that is allocated the highest number of CPUs, and how often this is obtained. The ultimate goal of the computational experiment is the determination of the optimal job run in terms of number of CPUs accessed and success of that access.

2. 3. Approach

As indicated above, the time available for the computation was set as six minutes. Once the job was allocated to the nodes, it was assumed that the job would have exclusive access to these resources and would run to completion within a pre-determined time frame. The pre-determined run time was set as two minutes.

Success of job execution was determined via a success criteria, where acquiring higher number of CPUs by a submitted job corresponded to a higher success criteria. In this experiment, this number was taken to be equal to the number of CPUs requested. In a more sophisticated setup, factors such as CPU power, network bandwidth, physical memory availability, storage access speeds, etc. will be taken into consideration for the success criteria.

In this experiment, what we call a "flooding" approach was followed, where jobs are rapidly submitted across several clusters simultaneously, each requesting different numbers of processors. Since all jobs take a total of two minutes of wall-clock time for completion, the higher the number of processors allocated, the higher the quality of the result. Therefore, the most desirable job is the one that obtains the highest number of processors, for this application.

Since a total of 6 minutes is available, it is actually possible for a job to 'wait' for four minutes in a queue before running, and still have success. During such a four-minute wait period, whenever a job begins execution, any other job with success criteria less than or equal to that of the current job, is cancelled, regardless of whether or not it had started execution. This ensures that resources are not wasted for jobs that are going to produce results of lower or equal quality, and thereby minimizes any adverse affects of the flooding approach.

The status of each job is queried continuously at five-second intervals between consecutive queries. Assuming the time between queries is negligible, there is only one job running at any given time.

At the end of each four-minute period, only the job with the highest success criteria would be running, which can then be cancelled, since the overall goal is the determination of the most successful job, rather than its output per se. Any other jobs that are then waiting for processors in the job scheduler queues are also cancelled.

We allocate resources in exclusive mode and the job is heavily CPU intensive. Therefore, it is highly likely that the job can be completed within the estimated two minute period, once the resources are allocated.

For example, suppose we submit jobs requesting 8, 16, 32 and 64 CPUs, respectively. When we query the status of the jobs, if 8- and 32-CPU jobs had started, we cancel the 8- and 16-CPU jobs, since we are guaranteed that the 32-CPU job would give a result. If the 64-CPU job also starts within the four-minute waiting period, we cancel the 32-CPU job as well and record that the best possible job was the 64-CPU job. Otherwise, we would record 32-CPU job as the best.

2.3.1. Undesirable affects of the flooding approach. The flooding approach provides a high probability of getting a job run. However, the approach could waste resources by running multiple instances of the same job. While we ensure that only one job is running at the end of each querying of the job status, the time interval between two queries of the jobs could be significant, especially if the login nodes of the clusters are heavily loaded, or there is a bad network connection between the job submission machine and the login nodes. During this time interval, multiple jobs might start running, where all but one job would be eventually canceled.

Since we are submitting multiple jobs requesting different numbers of CPUs, there is a possibility that the jobs compete with each other for CPUs. For example, suppose we submit 16-CPU and 32-CPU jobs to the same cluster and there are 35 CPUs idling at that moment. The best possible result would be if we could

get the 32-CPU job run. However, there is a possibility that the job scheduler would pick the 16-CPU job first, thereby preventing any possibility of the 32-CPU job to get in.

In order to determine the best running job at a given moment, it is required to check the status of each job frequently. Currently we are using simple SSH to obtain access into the login node to query the status, which results in large number of SSH commands on the login node.

2.3.2. Alternative approach. An alternative to the flooding approach would be to use an on-demand resource request/grant protocol, where a site will respond to inquiries about the maximum resource-provisioning that can be made at a given time. While this minimizes resource wastage, it may not be a practical solution in dynamic environments where the resource availability changes over time. For example, the resource availability during the resource acquisition phase could be less than that reported at the inquiry phase. Also, request/grant types of protocols are not supported at every computing site. Reserving resources would address some of these issues, but that would also involve some resource wastage. Currently most supercomputer centers do not provide user-settable reservation capability.

2.4. Test Environment

Table 1 lists the supercomputer clusters used for the experiment. We conducted several experiments with different sets of clusters selected each time. Table 2 provides the details of these experiments.

In experiment 1, all five clusters in Table 1 were used. In experiment 2, the TeraGrid SDSC was omitted, since this supercomputer was successful in getting most of the jobs running in the first experiment. In experiments 3, 4 and 5, we used the most heavily loaded clusters when individually run in isolation.

The job scheduler on DataStar normally takes about 15 minutes between two schedule attempts. Therefore, the last experiment was run with a higher interval between two queries of job status, in order to allow the job manager at least one chance to schedule the job on the DataStar.

In all instances, jobs were submitted requesting 8, 16, 32, 64 and 128 CPUs to each cluster except the TeraGrid at Argonne. It was only possible to submit the 8, 16, 32, and 64 CPU jobs on the TeraGrid cluster at Argonne.

3. Results

The term 'success rate' is used as the rate at which at least an 8-CPU job was able to run, and the term 'job submission' indicates one round of submitting jobs to each cluster. For each experiment, graphs are presented that indicate the success rate as a time varying function.

Table 1. Clusters used for the experiment.

Cluster	*CPUs*	*Architecture*
TeraGrid SDSC	512	IA-64, 1.5 Ghz
DataStar SDSC	1408	Power4, 1.5 and 1.7 Ghz
TeraGrid NCSA	1774	IA-64, 1.5 Ghz
TeraGrid ANL	124	IA-64, 1.5 Ghz
TeraGrid PSC	3000	Alpha EV68, 1Ghz

Table 2. Experiments conducted.

Experiments	*Clusters*	*Duration (hours)*
1	TeraGird SDSC, DataStar SDSC, TeraGrid NCSA, TeraGrid ANL, TeraGrid PSC	180
2	DataStar SDSC, TeraGrid NCSA, TeraGrid ANL, TeraGrid PSC	123
3	TeraGrid PSC	22
4	DataStar SDSC	22
5	DataStar SDSC	98

3.1. Experiment 1

For experiment 1, out of the 1464 job submissions, only 6 failed, giving a success rate of 99.59%. The 128-CPU job ran more than 50% of the times, and at least the 64-CPU job ran more than 80% of the times. Figure 1 gives the time varying behavior with six-hour intervals.

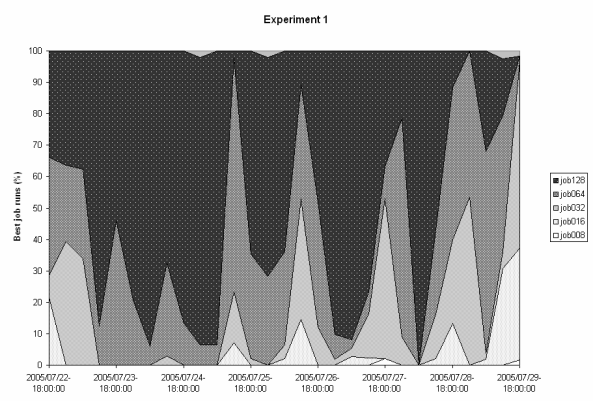

Figure 1. All clusters.

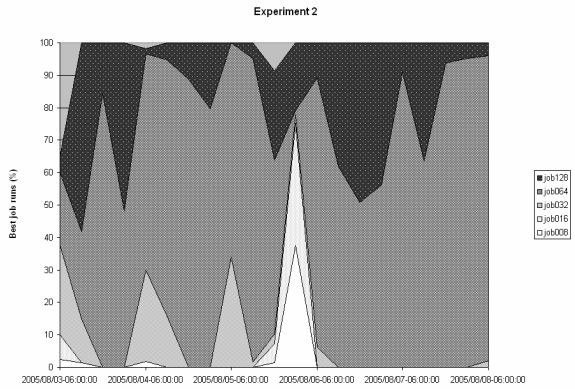

Figure 2. All clusters except TeraGrid SDSC.

3.2. Experiment 2

In experiment 2, the TeraGrid at SDSC was eliminated, and the overall success rate was 98.25% out of 1199 job submissions. However, the percentage of 128-CPU jobs reduced to just over 21%, while the percentage of getting at least the 64-CPU job remained over 80%. The time varying behavior for this experiment with six-hour intervals is given in figure 2.

3.3. Experiment 3

The Pittsburgh computer cluster is one of the heaviest loaded clusters, and only this cluster was used for experiment 3. As expected, the success rate dropped to 33.5%. The experiment was run for just over 21 hours during which 224 jobs were submitted. The time varying behavior with three-hour intervals is given in Figure 3.

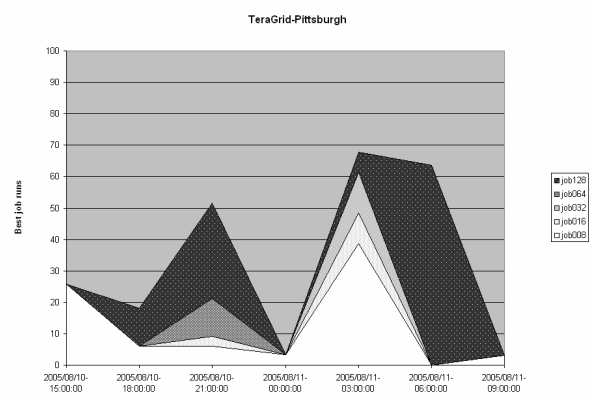

Figure 3. TeraGrid Pittsburgh.

3.4. Experiment 4

Experiment 4 included the other heavily loaded supercomputer, the DataStar at SDSC. During the 22-hour period when the experiment was performed, no job was able to run, giving a success rate of 0%.

While the main reason for job failures was the heavy load on this cluster, another major reason was the update frequency of the batch scheduler on DataStar. It was later identified that the interval between two consecutive job scheduling updates was around 15 minutes. Since there is only a four minutes wait time after submitting the jobs, there is a high chance that the scheduler does not get a single chance to attempt to schedule the submitted jobs. In order to verify this, this experiment was repeated after increasing the wait time to 20 minutes in experiment number 5.

3.5. Experiment 5

The main objective of experiment 5 was to determine whether the low frequency of scheduling on DataStar had any effect on the high rate of job failures.

For this experiment, the success rate was just under 50% with 277 job submissions. Even though we did not specifically record the scheduled times with respect to the time when a job actually began, we noticed that predominantly, the jobs started immediately after a scheduling cycle. Also, in many instances the scheduling was done after more than four minutes from job submission. Figure 4 gives the time varying behavior with one hour intervals.

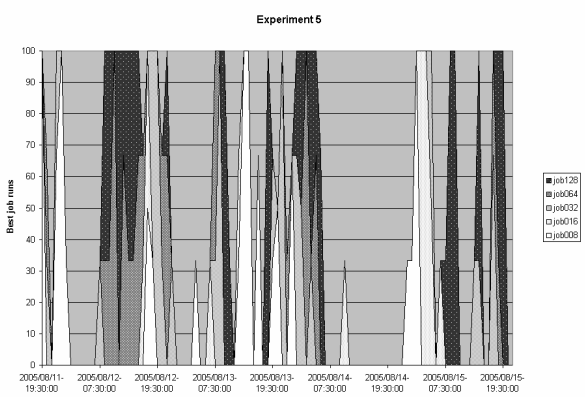

Figure 4. DataStar SDSC with 20 min wait.

3.6. Analysis

From the results of experiments 1 and 2, it can be easily concluded that, with a high number of TeraGrid clusters, the time critical neurosurgery simulation will be successfully run on some designated cluster with over 98% confidence. In the very small chance of run failure, the neuro surgeons would be able to rely on the local computers in the hospital, albeit sacrificing overall quality of images, since the local models are much more crude than those that can be performed on the supercomputer platforms.

In the case that heavily loaded clusters, such as the DataStar and the TeraGrid Pittsburgh, are used, it would be necessary to have multiple such clusters in order to obtain an acceptable success rate, as shown in 3.6.2. Since we had to increase the wait time on the DataStar to 20 minutes in order to obtain an acceptable success rate, one would be required to submit the job about 15 minutes in advance in a real neuro surgery application, and send the input data later when processors become available.

3.6.1. Backfilling. While the DataStar and the TeraGrid Pittsburgh were the most heavily loaded machines, other clusters also had many jobs in their job queues. However, the jobs submitted in these experiments were successfully scheduled to run before many of those jobs, mainly due to backfilling algorithms [5] used by schedulers.

The objective of backfilling is to make optimum use of the CPUs without compromising the start time of other jobs in front of the queue. This is accomplished by letting small jobs run in parallel with other larger jobs, if there is a sufficient number of CPUs for the small jobs. For example, consider a cluster having a total of 128 CPUs together with a 64-CPU job running for two hours. If the next job requires 128 CPUs, it can not begin until the first 64-CPU jobs completes. However, if a small job requesting 32 CPUs for 20 minutes is submitted, it can start without affecting the start time of the next 128- CPU job in front of the queue.

3.6.2. Clusters for acceptable performance. The number of clusters required to give an overall success rate can be estimated. We assume each cluster is equally loaded and therefore, has an equal probability of getting a job started.

Let an overall acceptable success rate be designated A, and the individual success rate of each cluster be S. Then, the failure rate of each cluster would be (1-S). With n clusters, the probability that the job will fail on all clusters would be $(1-S)^n$ and the probability that a job will run on at least one cluster would be $(1-(1-S)^n)$, which we want to be equal to A. Solving this equation gives,

$$n = \frac{\ln(1-A)}{\ln(1-S)}$$

For example, if the individual success rate is 20%, we would require 11 clusters to obtain an overall success rate of 90%.

3.6.3. CPU wastage due to flooding. While the flooding approach provides a higher probability of getting a job started within the critical time frame, there would be a certain amount of CPU waste due to less desirable jobs starting before, or at the same time, as the best job. The amount of this CPU waste would depend on the order that the jobs are started, and the time between two queries of job status, since at the end of each querying all the jobs except for the best one would be killed. An upper bound for the CPU waste can be estimated as follows.

Let $q\psi$ be the time between two queries, and $e\psi$ the job execution time (which was two minutes in our experiment). Also, let $n\psi$ be the number of clusters. It is assumed that the time to kill the less or equally desirable jobs is negligible.

The maximum waste would occur if all the jobs start immediately after a querying. Each job would run for q or e minutes, whichever is the minimum. Since we submit 8-, 16-, 32-, 64- and 128-CPU jobs to n clusters, the total CPU time utilized would be (8+16+32+64+128)*n*min(q,e). One of the 128-CPU jobs would be selected as the best job. Therefore, the wasted CPU time would be,

CPU_waste

= ((8+16+32+64+128)*n-128)*min(q,e)
= (248*n – 128)*min(q,e)

Since, one has little control over e, the only two variables that can be manipulated to reduce the waste are n and q. Minimizing either of them would lead to less CPU waste. However, decreasing n would decrease the probability of getting a job started. Reducing q too much would also have adverse affects, since the job status would be queried more frequently, resulting in adverse effects such as large number of SSH commands made on the login node. Therefore, it would be necessary to balance these variables carefully before running.

A lower bound for the CPU waste is zero, which would be the case when a 128-CPU job begins on some cluster initially, and no other job begins on any cluster until the subsequent querying.

4. Future work

In this work, only consideration of the number of CPUs was made to determine whether a particular job is more desirable than another. The CPU speed, available memory per CPU, etc, would also need to be taken into account for a more thorough analysis.

The waiting time for a job to start should depend on the quality of the network connection between the hospital and the remote cluster. If there is less bandwidth, the waiting time needs to be reduced to compensate for the extra time it would take to transfer data back and forth.

Even though the CPU waste has been minimized due to our flooding approach by killing less desirable jobs, it would be better to submit to a few clusters that have a higher probability of successful job start. One can use queue delay prediction algorithms [6] to select those clusters.

Public clusters have been used for the reported experiments. Another approach would be to use dedicated clusters. In the future, we envision a network of clusters, dedicated specifically for neurosurgery applications. In that scenario, another experiment would be to determine how a set of dedicated clusters would respond to many hospitals submitting jobs with the flooding approach.

As observed in the experiment with DataStar, if the scheduler's update frequency is very low, jobs would have to be submitted in advance and the input data sent later when they are available. In this scenario, should the program begin before the data is available, it will simply wait until the data arrives. The job execution time would need to be increased to accommodate this extra waiting time.

5. Acknowledgments

This work was supported in part by NSF ITR grants CNS 0427183, 0426558, IBM graduate student internship grant I3, and NIH grants P41 RR13218, P01 CA67165, R01 LM0078651. We would like to thank June Andrews, of UC Berkeley, for her summer internship work at SDSC analyzing data transfer performance across TeraGrid sites.

6. References

[1] A. Majumdar, A. Birnbaum, D. J. Choi, A. Trivedi, S. K. Warfield, K. Baldridge, and P. Krysl, "A Dynamic Data Driven Grid System for Intraoperative Image Guided Neurosurgey", *LNCS 3515*, 2005, pp. 672-679.

[2] S. K. Warfield, F. Jolesz, and R. Kikinis, "A High Performance computing Approach to the Registration of Medical Imaging Data", *Parallel Computing, 24,* 1998, pp. 1345-1368.

[3] P.M. Black, T. Morairty, E. Alexander, P. Stieg, E. J. Woodward, P. L. Gleason, C. H. Marting, R. Kikinis, R. B. Schwartz, and F. A. Jolesz, "The Development and Implementation of Intra-operative MRI and its Neurosurgical Applications Article", *Neurosurgery, 41*, 1997, pp. 831-842.

[4] M. Ferrant, A. Nabavi, M. Macq, F. Jolesz, R. Kikinis, and S. K. Warfield, "Registration of 3rd Intraoperative MR Images of the Brain Using Finite Element Biomechanical Model", *IEEE Transactions on Medical Imaging, 20, volume 12*, 2001, pp. 1384-1397.

[5] D. G. Feitelson, and A. M. Well, "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling", *12th Intl. Parallel Processing Symposium*, April 1998, pp. 542-546.

[6] J. Brevik, D. Nurmi, and R. Wolski, "Predicting Bounds of Queuing Delay for Batch-scheduled Parallel Machines", *Proceedings of ASM Principles and Practices of Parallel Programming*, March, 2006.