Novel Submission Modes for Tightly-Coupled Jobs Across Distributed Resources for Reduced Time-to-Solution

BY PROMITA CHAKRABORTY1,2, SHANTENU JHA1,2,3, DANIEL S. KATZ1
1Center for Computation & Technology, Louisiana State University, USA
2Department of Computer Science, Louisiana State University, USA
3e-Science Institute, Edinburgh, UK

The problems of scheduling a single parallel job across a large scale distributed system are well known and surprisingly difficult to solve. In addition, because of the issues involved with distributed submission like co-reserving resources, managing accounts and certificates simultaneously on multiple machines etc., the vast number of HPC-application users have been happy to remain restricted to submitting jobs to single machines. Meanwhile, the need to simulate larger and more complex physical systems continues to grow, with a concomitant increase in the number of cores required to solve the resulting scientific problems. One might reduce the demand on load per machines, and eventually the wait-time on queue by decomposing the problem to utilise two resources in such circumstances, even though there might be a reduction in the peak performance. This motivates the question: can otherwise monolithic jobs running on single resources be distributed over more than one machine such that there is an overall reduction in the time-to-solution? In this paper, we briefly discuss the development and performance of a parallel molecular dynamics code and its generalisation to work on multiple distributed machines (using MPICH-G2). We benchmark and validate the performance of our simulations over multiple input-data sets of varying size. The primary aim of this work however, is to show that the time-to-solution can be reduced by sacrificing some peak performance and distributing over multiple machines.

Keywords: Job submission paradigm, Tightly coupled distributed performance, Scheduling

1. Introduction

The need to simulate larger and more complex systems continues to grow, with a concomitant increase in the number of cores required to solve the resulting scientific problems. The number of processors typically available for specific compute jobs on a given machine has been increasing and will continue to increase. But often, problem sizes of interest are so large that they cannot be run on any available single resource. Additionally, utilising an increasing number of processors for a tightly-coupled simulation has its own challenges, as message-passing, the dominant paradigm for developing tightly-coupled applications, is reaching its limits of scalability – at least for some applications.

Significant effort continues to be invested in scaling-up the performance of applications. Critical as this endeavour is, there are limitations, as beyond some point, additional efforts will have diminishing returns. Not surprisingly, the metric of maximum concern to
the bulk of scientists is not peak performance (say measured in TFLOPS), but total time-to-solution ($T_s$). To a first approximation, the total $T_s$ can be decomposed into run-time ($T_{run}$), and wait-time for requested resources to become available ($T_{wait}$) (this is often just the queue wait-time). Ironically, although much attention has been focussed on reducing the $T_{run}$, little attention – at least compared to the former, is devoted to reducing the latter.

Jobs with larger processor count requirements are typically associated with longer wait-times than jobs with smaller processor counts. Additionally, the wait-time of a job in a queue depends on the estimated job-duration (maximum wall-clock-time requested), as well as the number of processors needed. At present, most users tend to submit jobs, however large, to a single machine, as opposed to a set of machines over distributed environment. This has two main advantages: (1) the job runs faster and (2) it consumes less CPU-hours than in distributed environment. The disadvantages are: (1) the job requires all resources to be available on a single machine, instead of distributing the load, (2) the job experiences larger wait-time in queue than when it is distributed across several machines.

Interestingly, although the dominant computational model for high performance computing has involved using only a single resource, with advances in Grid technologies, high-performance simulations over multiple resources are now feasible (6). In this case, decomposing a simulation to utilise two or more resources, is a natural response to the failure of a given system to meet peak demand; thus such decomposition can be referred to as an example of needful decomposition. However, users normally consider multiple machines only when the number of nodes required by the job is greater than what a single machine can provide; needful decomposition isn’t the only situation under which multiple, distributed resources can or should be used.

This motivates the question: can otherwise monolithic jobs running on single resources be distributed over more than one machine such that there is an overall reduction in the time-to-solution? What are the challenges in doing so? What is the performance? What system-level support is required to make such decomposition strategies meaningful/usable?

Although the capability to launch a single tightly-coupled task over two different resources exists, different workloads and queueing systems makes utilising distributed-job challenging in practise. As we will show in Section 3, tightly-coupled simulations, such as those using MPICH-G2, go into essentially a stalled state, waiting for all resources to become available. A commonly used approach is that of invoking the service of a co-scheduler, such as HARC (1) or GUR (2), and although the specific algorithms and implementation details of the co-schedulers differ, most co-schedulers require the prior permission of system-administrators to make a reservation. We propose a solution that, with some minor caveats, is essentially in user-space and does not require the pre-negotiation of advance reservation rights.

In this paper, we will demonstrate the feasibility of decomposition of a tightly-coupled simulation, to support novel submission modes as a strategy to reduce the effective time-to-solution; we refer to such decomposition as a type of opportunistic decomposition. We will briefly discuss the development and performance of a parallel molecular dynamics code and its generalisation to work on multiple distributed machines (using MPICH-G2). We benchmark and validate the performance of our application over multiple input data sets of varying size. The primary aim of this work however, is to show that jobs can finish sooner (i.e., lower time-to-solution) by an appropriate configuration of resource requests – defined as a selection of the number of processors and machines, even though the peak
performance might be poorer. Or formulated in a way that would make Zeno † proud: how to finish sooner by running slower!

The outline of this paper is as follows. In Section 2, we describe our parallel MD code, the models and testbeds used. In Section 3 we narrate and depict the control flow for job submission to a single machine as well as multiple machine(s), and highlight the difference in environments. In Section 4 we present results – on job-runs, actual wait-times taken, wait-time predictions from BQP, formulate the necessary equations, and show that often the time-to-solution is lower in the distributed mode, in spite of higher run-time values. Our conclusions and some discussion to other work is presented in Section 5.

2. Tightly-Coupled Simulations: Background Information

Due to the frequent communication requirement, MD codes are typically deployed on tightly-coupled machines. It is often difficult to scale message passing codes to beyond O(1000) process. Inter-process communication bandwidth requirements are not very large, but MD codes are latency sensitive. Thus, although MD simulations are not naively suitable for distributed parallelism, our aim is to see if new ways of “distributing” such applications can yield better performance, not only in terms of speed, but also in terms of other issues such as queue wait-time, load balance, etc., and to see if current grid implementations support this paradigm.

We developed a parallel molecular dynamics (MD) code (5) using MPICH-G2 and C++, based on the serial MD code Mindy †. Our domain decomposition was not complex; we simply divided the dataset and the number of atoms amongst the available processors. We studied the performance of this parallelized and distributed molecular dynamics code initially on LONI (Bluedawg, Zeke, Ducky), and then on the TeraGrid (NCSA TeraGrid Cluster, and SDSC TeraGrid Cluster) machines. LONI is a Louisiana-wide network of supercomputers connected by light-paths, and is connected to the TeraGrid (TG). NCSA and SDSC TG Clusters are IA-64 machines.

We used the Batch Queue Predictor (BQP) (3) to get an estimate of the wait-time of jobs on NCSA and SDSC TG clusters. Currently, BQP tools provide two types of estimates for a given set of job characteristics: (1) they can estimate a statistical upper bound on job wait-time in the queue prior to execution, (2) given a start deadline, they can calculate the probability that the job begins execution by the deadline. We used the second option. It is to be noted that BQP predictions are for single machines only.

In order to avoid confusion and to retain clarity, we define some terminology that we used in this paper. By job submission to a single machine, we mean that the user specifies that the whole job is to be submitted to a specific single machine (stand-alone) available on the grid. By job submission to multiple machines we mean that the user specifies that a single job is to be divided (in sub-jobs) among a particular set of machines available. Thus a sub-job is a part of a single job that is submitted to one of a set of machines, not a smaller piece of job submitted to a node of a machine. Time-to-solution, T2S, (T_s) is a measure of the sum of run-time and queue wait-time for a job.

3. Control Flow For Job Submission: Single And Multiple Machines

Often, a job is remotely submitted through Globus, using an RSL (Resource Specification Language) submission file that contains information about the specific machine(s) for job

† An ancient Greek philosopher who formulated paradoxes that defended the belief that motion and change are illusory
† Available from the NAMD website http://www.ks.uiuc.edu/namd

Article submitted to Royal Society
submission, the number of processors needed, the maximum wall-clock-time, etc. Once
the job is submitted, control passes to GRAM. The RSL file is parsed to get the resource(s)
information. The GRAM-gatekeeper interacts with the job schedulers and submits the jobs
to the respective scheduler(s). Some examples of job schedulers are LoadLeveler (on the
LONI IBM P5 clusters), and PBS (on Queen Bee). If a job is submitted to a single machine,
the local scheduler adds the job to the queue where it waits until requested nodes becomes
available. The local resource manager (RM) assigns the available processors to the waiting
jobs, often using FIFO or other fairness principles. When an MPI job starts executing,
an MPI environment/communicator is created and initialized, with N processes. If the job
does not finish execution within the specified wall-clock-time value provided by the user
in the RSL file, the RM kills it. So, users should have a good idea of the time the job might
take to execute. On the other hand, if the wall-clock-time value provided is too large, the
jobs may stay in the queue longer.

Figure 1: (a) Control flow for job submission to a single machine; the dotted rectangle represents a
single machine. MPI/MPICH-G2 creates a MPI Communicator World to keep track of the communication
between different processors in the machine. (b) Control flow for job submission to multiple
machines; each dotted rectangle represents a single machine. By communicating with the RM and
GRAM, the MPICH-G2 environment ensures that the sub-jobs on all the distributed machines are
allocated the required number of processors.

If a job is submitted to multiple machines using a single submission (with a single
RSL file), the process is more complicated. The code needs to be compiled using MPICH-
G2. The GRAM-gatekeeper interacts with all the machines requested by the user, and the
respective job schedulers put the sub-jobs in the respective queues. Here, the RMs might
not be able to allocate processors to all the sub-jobs simultaneously. When the first set of
resources is allocated, the sub-job in that machine initializes the MPI-environment, and
Table 1: Performance on single machines using 8 processors in each case.

(a) Time taken by BL & DU (BrH)

<table>
<thead>
<tr>
<th>Timesteps</th>
<th>$T_r$(BL)</th>
<th>$T_r$(DU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>1,000</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>10,000</td>
<td>23.9</td>
<td>23.9</td>
</tr>
<tr>
<td>100,000</td>
<td>232.9</td>
<td>235.3</td>
</tr>
<tr>
<td>1,000,000</td>
<td>2518.6</td>
<td>-</td>
</tr>
</tbody>
</table>

(b) Time taken by BL, DU & ZE (Alanin)

<table>
<thead>
<tr>
<th>Timesteps</th>
<th>$T_r$(BL)</th>
<th>$T_r$(DU)</th>
<th>$T_r$(ZE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>5.6</td>
<td>5.6</td>
<td>6.0</td>
</tr>
<tr>
<td>100,000</td>
<td>11.1</td>
<td>11.2</td>
<td>12.2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>110.6</td>
<td>111.9</td>
<td>121.1</td>
</tr>
<tr>
<td>10,000,000</td>
<td>1098.7</td>
<td>1113.97</td>
<td>1207.0</td>
</tr>
<tr>
<td>100,000,000</td>
<td>-</td>
<td>10860.1</td>
<td>11054.9</td>
</tr>
</tbody>
</table>

waits for responses from other machines. While the sub-job is waiting for other sub-jobs to start, it is officially in a run state from the point of view of the local RM, and if it waits for too long, it will be killed. Hence, it is important that the sub-jobs in all the different machines are started simultaneously. Figures 1(a) and 1(b) show the control flow for job submission to a Grid for a single machine and for multiple machines respectively.

4. Results and Discussions

We used two physical systems as models for our experiments: Alanine (alanin), and Bacteriorhodopsin (BrH). Alanin is a 66 atom polypeptide, and BrH is a crystal structure of bacteriorhodopsin containing 3762 atoms.

Distributed MD performance results on LONI: Initially, our aim was only to test the feasibility of running a single parallel job across multiple machines, and to quantify the performance of the job. We executed the parallel Mindy code on three IBM AIX P5 clusters, Bluedawg (BL), Ducky (DU), and Zeke (ZE), on LONI. Here, we provide the data and a quantitative analysis for it, and we estimate the performance degradation associated with distributed job submission. On LONI the wait-times for our jobs were trivial – of the order of a few minutes (although there were cases when jobs were killed while waiting in queue for a long time, due to the system being overloaded). Hence, we concentrated on run-time, $T_r$, i.e. CPU-seconds used, only. The total amount of CPU time consumed for a simulation was calculated using the formula: CPU-secs = time taken × Number of processors, where time taken is measured in seconds. Tables 1(a) and 1(b) highlight the application’s performance on individual LONI machines; Table 1(a) shows the performance on BL and ZE separately when the code runs on eight processors (on a single node) using the BrH model. Table 1(b) shows the same for Alanin model, with machines BL, DU, ZE. The tables show that the processing time is proportional to the number of timesteps over which the simulation is run. Thus, at the largest timestep values, it is valid to assume that initial/transient/set-up effects are no longer relevant. The tables also show that the performance of BL and DU are essentially the same, to within a few percent.

Having established that start-up effects no longer contribute at $10^5$ timesteps, we measured the computational time required ($T_r$) as the processor count was varied on DU (Table 2). As shown in the table, for the BrH model there is a speed-up as the processor count is increased up to 24 (although not a slope of one); after which increasing the processor count leads to a net increase in computational cost.

Table 3 compares the computational time (CPU-secs) for the BrH model when run on DU (stand-alone) with the computational time taken when run on DU and ZE (combined)

Article submitted to Royal Society
Table 2: Performance comparison as measured by the total computational time (CPU-secs required) on Ducky when using different number of processors for $10^5$ and $10^6$ timesteps (for BrH model)

<table>
<thead>
<tr>
<th>Px</th>
<th># Nodes</th>
<th>$100,000$ Timesteps</th>
<th>$1,000,000$ Timesteps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$T_r$ (sec)</td>
<td>CPU-secs</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>118.8</td>
<td>1426.1</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>46.8</td>
<td>1123.7</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>36.7</td>
<td>1175.5</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>32.5</td>
<td>1299.6</td>
</tr>
<tr>
<td>48</td>
<td>6</td>
<td>29.6</td>
<td>1420.0</td>
</tr>
<tr>
<td>56</td>
<td>7</td>
<td>28.7</td>
<td>1606.1</td>
</tr>
</tbody>
</table>

*Notations: Px = Number of processors; $T_r$ = Run-time;

Table 3: A comparison of the time taken (in seconds) by DU alone, with the time taken on DU & ZE (combined) for different processor configurations (for BrH model)

<table>
<thead>
<tr>
<th>Timesteps</th>
<th>DU</th>
<th>DU+ZE</th>
<th>PD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Px</td>
<td>$T_r$ (s)</td>
<td>CPU-secs</td>
</tr>
<tr>
<td>1,000</td>
<td>8</td>
<td>2.4</td>
<td>19.5</td>
</tr>
<tr>
<td>10,000</td>
<td>23.9</td>
<td>191.5</td>
<td>1882.7</td>
</tr>
<tr>
<td>100,000</td>
<td>76.2</td>
<td>123.7</td>
<td>1219.0</td>
</tr>
<tr>
<td>1,000</td>
<td>16</td>
<td>0.7</td>
<td>11.1</td>
</tr>
<tr>
<td>10,000</td>
<td>7.7</td>
<td>123.7</td>
<td>1219.0</td>
</tr>
<tr>
<td>100,000</td>
<td>36.7</td>
<td>1175.5</td>
<td>1282.7</td>
</tr>
</tbody>
</table>

*Notations: PD = Performance Degradation; Px = Number of processors; $T_r$ = Run-time;

for a range of different processor count configurations. We see that the performance in the distributed environment degrades between 2-22% (approx.) compared to the performance on a single machine. Further, increasing the number of processors, the performance improves and the degradation percentage decreases (see Figures 2(a) and 2(b)). This raises interesting possible usage modes, i.e., instead of spending extra hours in queue waiting for resource allocation, whether parallel code should be distributed over multiple machines.

**Distributed MD performance results on TeraGrid with emphasis on wait-time:**
After testing on LONI, we performed similar experiments on the TeraGrid. But unlike on LONI (where wait-times were often negligible), here we also recorded the wait-time taken by the jobs. The TeraGrid environment proved to more challenging than LONI due to the size of the virtual organization and multiple resource providers with separate policies; LONI resources used for this experiment, in contrast, were essentially identical – both in terms of policy and infrastructure.

Table 4 compares the computational time (in secs.) for the BrH model, when run on NCSA and SDSC Clusters (stand-alone), with the computational time taken when run on both the NCSA and SCSC Clusters (combined), for a range of different processor count configurations.
Novel Submission Modes . . . for Reduced Time-to-Solution

![Graphs showing time to solution for different configurations](image)

(a) 16(DU) vs. 8+8(DU+ZE) processors  
(b) 32(DU) vs. 16+16(DU+ZE) processors

Figure 2: A performance comparison of the time taken (in seconds) by single machine versus two machines at a time (for BrH) on LONI.

Table 4: A comparison of the time taken (in seconds) by TeraGrid machines NCSA Cluster & SDSC Cluster for different processor configurations (for BrH model)

<table>
<thead>
<tr>
<th>Timesteps</th>
<th>NCSA</th>
<th>SDSC</th>
<th>NCSA + SDSC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Px</td>
<td>T_r</td>
<td>T_w</td>
</tr>
<tr>
<td>1,000</td>
<td>8</td>
<td>3.67</td>
<td>306</td>
</tr>
<tr>
<td>10,000</td>
<td>35.77</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td>352.85</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>16</td>
<td>3.01</td>
<td>1050</td>
</tr>
<tr>
<td>100,000</td>
<td>27.33</td>
<td>1062</td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td>240.96</td>
<td>1062</td>
<td></td>
</tr>
</tbody>
</table>

*Notations: Px = Number of processors; T_r = Run-time; T_w = Wait-time;

configurations. We observe that the run-time of jobs is proportional to the number of timesteps calculated, as on LONI. Roughly, for a tenfold increase in timesteps, the run-time for the jobs increased tenfold. But, the most interesting thing to note is the behavior of the wait-time. Though the wait-times remained more or less constant for increasing timesteps (i.e. for increased run-times), they increased dramatically for increasing number of processors. Moreover, jobs run on NCSA and SDSC combined, had significantly lesser wait-time than those run on either alone. Hence, we concluded that an overall reduction in the time-to-solution in distributed mode is feasible, in spite of the fact that stand-alone machine run-times were far less than in the combined (distributed) scenario.

**Opportunistic Distributed Resource Usage (BQP):** For single resources, the Batch Queue Predictor (3) (BQP) provides users with the probability of running a job on a number of nodes within a certain pre-selected deadline. As BQP predictions are for single machines only, we first estimated the wait-time in queue separately for each of the machines involved using the BQP tool. We then combined the results to get the approximate wait-time for the two machines – the NCSA and SDSC TG Clusters (see Table 5). We then used this as estimated average wait-time, in order to attempt to reduce the total time-to-solution over multiple resources. The use of BQP keeps the entire scheduling decision and resource co-allocation in user space (as opposed to when using advanced reservations).

*Article submitted to Royal Society*
Table 5: BQP Data for NCSA and SDSC TeraGrid Clusters (for Queue=dque for both). The BQP user provides the value of the number of processors to be used and a best-estimate for the run-time required; based upon that BQP returns a value of the wait-time on a resource for different probabilities. We choose a value of approximately 0.6; our results are not sensitive to the specific value of probability chosen.

<table>
<thead>
<tr>
<th>Px (Number of processors)</th>
<th>Estimated $T_r$ (seconds)</th>
<th>Probability</th>
<th>Approx. $T_w$ (NCSA) (seconds)</th>
<th>Approx. $T_w$ (SDSC) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>900</td>
<td>0.61</td>
<td>530</td>
<td>780</td>
</tr>
<tr>
<td>1800</td>
<td>2130</td>
<td>0.61</td>
<td>4200</td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td>2130</td>
<td>0.61</td>
<td>4200</td>
<td></td>
</tr>
<tr>
<td>7200</td>
<td>5150</td>
<td>0.61</td>
<td>8100</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>900</td>
<td>0.61</td>
<td>2130</td>
<td>4200</td>
</tr>
<tr>
<td>1800</td>
<td>2130</td>
<td>0.61</td>
<td>4200</td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td>5150</td>
<td>0.61</td>
<td>8100</td>
<td></td>
</tr>
</tbody>
</table>

*Notations: Px = Number of processors; $T_r$ = Run-time; $T_w$ = Wait-time; (times measured in secs.)*

Table 5 shows the estimated wait-time as calculated by BQP for the NCSA and SDSC TG Clusters. We see that in some cases, higher run-times do not cause the wait-times to increase. This was also supported by actual run results in Table 4, where the wait-time does not increase with the number of timesteps. Now, if we consider jobs with run-time of 900 seconds from Table 5, we see that for 8 processors the approximate wait-time is 530 seconds; we got the same value for 4 processors (not shown here); but for 16 processors, the wait-time jumps to 2130 seconds. Hence, if we distribute this job into two 8 processor sub-jobs across the two machines, the wait-time is reduced by a factor of four, though the run-time might increase by 10-20% as was observed earlier (Tables 3 and 4). So, the net time-to-solution decreases in the distributed mode in such cases.

Overall, in designing a multiple-machine job submission paradigm, BQP prediction can help us in two ways. First, to help us decide whether it is beneficial to distribute (as in the case discussed above), and secondly, if beneficial, it tells us what the expected wait-time in the distributed paradigm is.

(a) Quantitative Decision-Making

Let $T_{wait}(i, q)$ be the wait-time and $T_{run}(i, q)$ the run-time for a job on machine $i$ using $q$ processors. If $T_s(i, q)$ is the estimated T2S of a job on machine $i$ using $q$ processors, then $T_s(i, q) = T_{wait}(i, q) + T_{run}(i, q)$. For jobs spread over multiple machines: $T_s(i, q_1, q_2, ..., q_k) = \max[k, q_k] + T_{run}(k, q_k), k \in \{i, j, \cdots\}$, where the $\max$ is computed over all machines used.

We have obtained wait-time in two different ways: directly from job runs, and from BQP data. BQP provides an estimate of the $T_{wait}(i, q)$ which varies not only quite significantly with $i$ (i.e., different resources have different load-levels at a given time), but also can vary significantly for different values of $q$ for the same $i$ (for example the $T_{wait}(mymachine, 16) << T_{wait}(mymachine, 32)$). Thus by estimating the values $T_{run}(i, q)$, the optimal values of $i$ and $q$ can be determined so as to find the minimum $T_s$.

Let, for the single machine $i$ (either NCSA or SDSC) using 16 processors, $T_s(i, 16) = T_{run}(i, 16) + T_{wait}(i, 16)$, and for the two machines, let, $T_s(NCSA, 8; SDSC, 8) =$
$T_{\text{run}}(\text{NCSA}, 8; \text{SDSC}, 8) + T_{\text{wait}}(\text{NCSA}, 8; \text{SDSC}, 8)$. From data collected on TerraGrid and LONI, we obtained,

$$T_{\text{run}}(i, 8; j, 8) \approx aT_{\text{run}}(i, 16), \text{where, } 1.1 \leq a \leq 1.2,$$

for 10-20% performance degradation, where $a$ is a measure of performance degradation in switching from one machine to multiple machines.

Based upon empirical (reproducible) observation on the TG we find:

$$T_{\text{wait}}(i, 16) >> T_{\text{wait}}(\text{NCSA}, 8; \text{SDSC}, 8)$$

which often holds true for a range of processor counts, and $i$ could be either the NCSA or SDSC IA64. For example, for an estimated run-time of 900 secs. for 16 vs. 8+8 processors:

$$T_{\text{wait}}(\text{NCSA}, 8; \text{SDSC}, 8) \approx bT_{\text{wait}}(i, 16)$$

where $b$ is a measure of the relative wait-times for one machine versus multiple machines and is empirically about 0.37 for NCSA machines and 0.19 for SDSC’s IA64. This derives from data for wait-times for NCSA, which are 530s & 2130s – and that for SDSC are 530s & 4200s for 8 & 16 processors respectively, whilst for NCSA+SDSC the maximum wait-time is 780s. Hence, overall, time-to-solution,

$$T_s(\text{NCSA}, 8; \text{SDSC}, 8) = T_{\text{run}}(\text{NCSA}, 8; \text{SDSC}, 8) + T_{\text{wait}}(\text{NCSA}, 8; \text{SDSC}, 8)$$

For distribution to be beneficial (ideal case),

$$0.2T_{\text{run}}(\text{NCSA}, 16) - 0.63T_{\text{wait}}(\text{NCSA}, 16) \leq 0$$

or,

$$T_{\text{run}}(\text{NCSA}, 16) \leq 3.15T_{\text{wait}}(\text{NCSA}, 16)$$

Thus for all future jobs, with a predicted wait-time of $T_{\text{wait}}(\text{NCSA}, 16)$, we can safely distribute those jobs for which the expected $T_{\text{run}}(\text{NCSA}, 16)$ is 3.15 times $T_{\text{wait}}(\text{NCSA}, 16)$. However, in practical world, we may have situations where,

$$0.2T_{\text{run}}(\text{NCSA}, 16) - 0.63T_{\text{wait}}(\text{NCSA}, 16) > 0$$

Thus, it is often a trade-off between acceptable wait-time versus additional CPU-hours spent. It is important to point out that there are fluctuations due to changing work and queue-loads in the values of the various parameters, e.g., $a \& b$, and $T_{\text{wait}}$; however, the general results hold irrespective; specifically, BQP result factors in these fluctuations in its probabilistic predictions.

5. Conclusion

In this paper, we have highlighted the need for and advantages of distributed job submission. We started from a simple, sequential MD code, parallelised it, and extended it to run over distributed resources using MPICH-G2. Importantly, we showed that when running
over multiple machines, even when simulating relatively small physical models, the performance is comparable to when running on a single machine. This formed the basis for the next phase of our work: investigating submission modes for a tightly-coupled simulation in order to reduce the $T_s$. Our approach was to circumvent the static model of fixing the number of processors on a pre-determined set of resources using a co-scheduler; in contrast we adopted an agile-execution model, where we determined the best resource and configuration to use almost at run-time.

We also have shown that, on average, the wait-time for a job submitted to a single machine can be longer than the wait-time of the same job when decomposed and submitted to multiple machines. We postulate that this is due to the typically lower resource requirement per machine. When combined with sophisticated tools such as the BQP, which provides a good estimate to a first approximation of the probability of a job to run in a given window, opportunistic decomposition has the clear potential to increase throughput — to significantly lower wait times with relatively insignificant increase in CPU usage. We contend that this is the first documented work to use a prediction tool such as BQP at deployment (i.e. just before runtime) to effectively decompose tightly-coupled simulations.

It is important to note that such opportunistic scheduling is not guaranteed to be successful; distributing over multiple machines will not always lower wait-times, as it is conceivable that the wait-time for any one of the small jobs on a machine could be larger than the wait-time for a single simulation on a larger machine, if not indefinitely long in the pathological case.

Although we show a reduction in the $T_s$ for a tightly-coupled simulation decomposed over multiple machines, our approach is sufficiently general that we can decompose a large job into smaller jobs even on a single machine and still find a reduced $T_s$. Hence, the claim that this is a novel submission mode. And it is worth mentioning that our approach is also valid for ensembles of simulations, including high-throughput computing, where optimal aggregation strategies of “small jobs” into “big jobs” can be made; in some ways, the aggregation approach is the reverse of the decomposition approach, but it can be implemented across different machines. We will report results on this in a future paper.

This work is a part of the Cybertools project (4) and is supported by NSF/ EPSCoR Award No. EPS-0701491. The authors would like to thank Wei Huang for help in generating large test models.

References


Article submitted to Royal Society