Data Staging Effects in Wide Area Task Farming Applications *

Wael R. Elwasif †‡ James S. Plank ‡ Rich Wolski ‡

elwasifwr@ornl.gov, [plank,rich]@cs.utk.edu

Abstract

Recent advances in computing and communication have given rise to the computational grid notion. The core of this computing paradigm is the design of a system for drawing compute power from a confederation of geographically dispersed heterogeneous resources, seamlessly and ubiquitously. If high-performance levels are to be achieved, data locality must be identified and managed. In this paper, we consider the effect of server side staging on the behavior of a class of wide area "task farming" applications. We show that staging improves task throughput mainly through the increased parallelism rather than the reduction in overall turnaround time per task. We derive a model for farming applications with and without server side staging and verify the model through live experiments as well as simulations.

1. Introduction

The computational grid [9] has recently evolved into a powerful paradigm for the utilization of distributed computational resources. Projects such as Globus [8], Legion [10, 11], NetSolve [4], Condor [7, 16], Ninf [12, 14] and EveryWare [17] are but a few of the many projects that attempt to harness the power of the computational grid. By definition, the computational grid encompasses resources that span a wide geographical (and network) distance. As such, the issue of data locality plays an important role in enhancing the performance of distributed applications running on the grid.

Locality has always been an important element in reducing data access overhead, whether at the processor level or in a distributed setting. While it is generally accepted that staging data near where it is consumed (caching) improves general application performance, the nature of such an improvement depends on the type of the application and on its ability to effectively use the available data.

Parameter sweep (or farming) aplications is a class of applications with attractive scalability properties, which renders it suitable for grid implementation. These embarrassingly parallel applications divide a (potentially huge) parameter space into regions. A worker application is then assigned the task of searching one such region for points that match the search criteria set by the application. A subclass of such applications are *data independent parameter sweeps*, where the input data set is shared between all workers. The structural properties of this sub-class suggest the use of compute-side data staging to reduce input data fetching communication delay. The resulting reduction in overhead should increase application throughput (expressed in terms of the number of regions searched per unit time).

In this paper, we study the staging behavior of wide-area data-independent parameter sweep applications. We base our computing model on NetSolve [4], and our staging using the IBP storage system [13]. We first derive an analytical model for predicting performance in such systems, and then run experiments in a wide-area setting to determine how well the model fares in a real implementation. Finally, we show results of a simulation of these systems so that a wider class of parameters can be analyzed.

From this work, we conclude that in these applications, staging improves throughput by reducing the purely sequential portion of the overall application through the overlapping of the data transfer signal of a task with the initiating control signal of subsequent tasks. This increases job initiaition rate and application parallelism. Our analytical model shows that this improvement could be adversely affected by the computational servers' sharing of bandwidth to the staging point, which increases data transfer time upto the point where the addition of more compute servers can lead to deterioration in application performance.

*This material is based upon work supported by the National Science Foundation under grants ACI-9876895, EIA-9818334 and EIA-9975015, the Department of Energy under grant DE-FC0299ER25396, and the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

†Distributed Computing Group, Oak Ridge National Lab Oak Ridge, TN 37831, USA

‡Department of Computer Science, University of Tennessee, Knoxville, TN 37996, USA

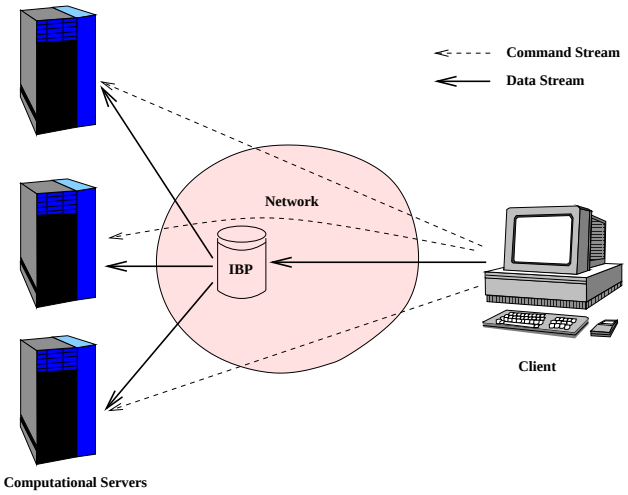

Figure 1. Farming model with an IBP Staging Point

2. Applications

We focus on parameter sweep, or "farming," applications, which have been the focus of much initial work on grid computing [1, 2, 5, 6, 15]. As described above, these are parallel applications where little or no communication is performed between sub tasks in a larger computational process. An example is MCell, a microphysiology application that uses 3-D Monte-Carlo simulation to study molecular bio-chemical interactions within living cells [6]. In fact, almost all Monte-Carlo simulations belong to this class of applications.

The applications that we study have the following characteristics:

- **Large input data set.** The input data set is large relative to any control parameters that determine the manner in which the data set is processed.

- **Large number of tasks.** The number of tasks is large and is not usually known in advance. The application sweeps through a (possibly infinite) parameter space defined through the control parameters. Task throughput is the paramount measure of overall application performance.

- **Task independence.** Each task is completely defined through its control data set and the global input data set. No dependencies exist between tasks in the parameter sweep application.

3 Computing Model

We adopt the computing model supported by NetSolve [4]. This is a brokered RPC model, sometimes termed the "client-agent-server" model. A client submits problems to computational servers through an intermediate agent, which acts as a resource broker and performance monitor for all computational servers that are registered with this agent. The client running the application queries the agent for a list of servers capable of solving the problem at hand. The client then proceeds to submit instances of the parameter sweep application to servers in that list until no servers are left unused. The client then proceeds to monitor completion of tasks on active servers, submitting more tasks to the server(s) that complete their tasks.

The introduction of staging into this environment leads to the differentiation of the control and data streams within the application. Without staging, the client marshals all inputs to the computational problem to the server whenever an instance of the problem is instantiated. With staging, the input data is prepositioned on a storage server, perhaps near the computational servers, and a pointer to the data is sent to the servers on task instantiation. The servers retrieve the data from the storage after their tasks are instantiated. This separates the path of the control data from that of the (larger) input data as seen in Figure 1. We assume that network storage exists for staging, such as the storage provided by the Internet Backplane Protocol (IBP) [2].

To analyze the effect staging has on the throughput in a parameter sweep application, we make the following assumptions:

- Pure execution time of individual tasks on computational servers is constant. Moreover, each server executes one task at a time.

- Tasks are introduced into the computational environment one at a time. The reasoning behind this assumption is twofold. The simultaneous introduction of tasks from the client side to multiple computational servers imposes heavy demands on bandwidth for problems involving large data sets. Such usage of available bandwidth may not be acceptable to many organizations. The second reason is that in some environments (e.g. NetSolve), the introduction of a problem instance into the environment alters the configuration in such a way as to affect the decisions involved in assigning future instances to servers. While devising a scheme for the simultaneous scheduling of multiple problem instances is feasible, it is beyond the scope of this research.

- No server failures occur during computations.

- The output data size is negligible. It follows that no significant time elapses between a task terminating on a server and the client's recognition of this event and the launch of another instance on the same server.

In analyzing the throughput of parameter sweep applications, we define the following variables:

- T_f: Task forking time. This is the time to select a computational server and start a process on it. It includes network latencies and protocol overhead, but not the time to transmit input data.
- T_t: Input data transmission time.
- T_c: Task computational time, once the input has been received.
- T: A sufficiently large time interval used to measure throughput.
- P: Total number of identical computational servers.
- P_e: Effective number of computational servers.
- N: Number of completed tasks in T seconds.

With no staging, tasks are started as depicted in Figure 2(a). The task turnaround time is $(T_f + T_t + T_c)$ *Sec/Task*. Since a new task cannot be started until all data from the previous task has been transmitted, the task initiation rate is $1/(T_f + T_t)$ *Tasks/Sec.*. It follows that, the maximum number of tasks that can be started before the first task terminates is given by

$$P_e = (T_f + T_t + T_c)\,\frac{1}{T_f + T_t} \quad Tasks/Finished\ task$$

A new task initiated after completion of the first task does not increase the number of simultaneously active servers. By starting this new task on the same server where the recently terminated task was assigned, we get the scenario depicted in Figure 2. It should be noted that this choice is possible because of the assumption that all servers are identical, which makes it immaterial which server gets to process any given task. It follows that, *regardless of the number of available servers*, the maximum number of servers that can be kept *simultaneously* busy (P_e, for "effective" number of servers) depends on $(T_f + T_t)$. In Figure 2(a), P_e equals three. Assuming $P_e \leq P$, then

$$\begin{aligned} N &= P_e\,\frac{T}{T_f + T_t + T_c} - (P_e - 1) \\ &= \frac{T - T_c}{T_f + T_t} \end{aligned} \tag{1}$$

With staging, we add primes to all variables. Note that $T_f' = T_f$ and $T_c' = T_c$. However T_t' will differ from T_t, depending on the location of the staging point. As depicted in Figure 2(b), a task need not wait for its input before starting, but rather, begins fetching its inputs from the staging

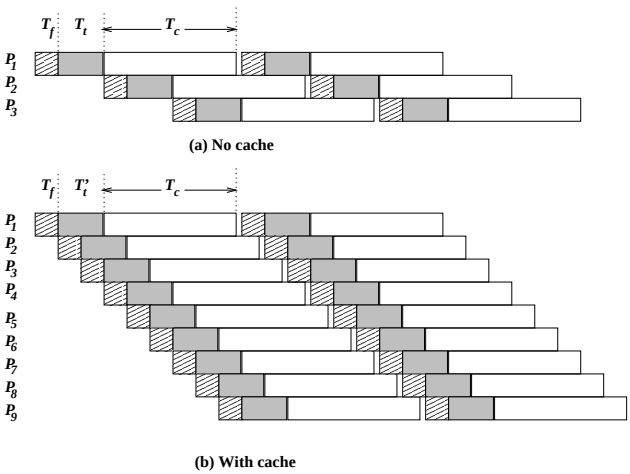

Figure 2. Task launching in parameter sweeps

point as soon as it is initiated. The result is that tasks are able to fetch their inputs in parallel up to the point where the bandwidth consumed by all tasks in their fetch phase is equal to the capacity of the network link. This increases the effective number of servers (to nine in Figure 2(b)), and will also shorten task turnaround time if $T_t' < T_t$. It follows that

$$P_e' = \frac{T_f + T_t' + T_c}{T_f} \quad Tasks/Finished\ task$$

And assuming $P_e' \leq P$, then

$$\begin{aligned} N' &= P_e'\,\frac{T}{T_f + T_t' + T_c} - (P_e' - 1) \\ &= \frac{T - T_t' - T_c}{T_f} \end{aligned} \tag{2}$$

From equations 1 and 2

$$\frac{N'}{N} = \frac{T - T_c - T_t'}{T - T_c}\,\frac{T_f + T_t}{T_f} \tag{3}$$

For sufficiently large T, $T - T_c \gg T_t'$. Then

$$\frac{N'}{N} \approx 1 + \frac{T_t}{T_f} \tag{4}$$

Equation 4 shows that the improvement we see from the use of staging does not depend on transmission time (T_t') up to the point of link saturation.

The staging performance modeled above assumes the number of available servers exceeds the maximum number of effective servers that a client can use, and that the slowdown of input transfer caused by link saturation is negligible. The improvement in performance is due to the reduction in transmission time and the enhanced parallelism

achieved through overlapping task instantiation and data transmission. To isolate the effect of staging, let $P < min(P_e, P'_e)$. In this scenario, we find that

$$N = P \frac{T}{T_f + T_t + T_c} - (P-1) \quad (5)$$

$$N' = P \frac{T}{T_f + T'_t + T_c} - (P-1) \quad (6)$$

For sufficiently large T, $P-1 \ll P \frac{T}{T_f + T_{T_*} + T_c}$. It follows that

$$\frac{N'}{N} \approx \frac{T_f + T_t + T_c}{T_f + T'_t + T_c} \quad (7)$$

For a server side staging point, we have $T'_t \ll T_f + T_c$. Then

$$\frac{N'}{N} \approx 1 + \frac{T_t}{T_f + T_c} \quad (8)$$

For a client side staging point, $T'_t = T_t$ and $N'/N \approx 1$. In the vast majority of scientific farming applications, the computational time T_c dominates data transmission time T_t. This is in contrast to web caching applications, where the opposite is generally true. Hence, it can be seen from the above model that as the number of computational servers available decreases from that number needed to keep the staging pipeline full, the throughput gain advantage through staging diminishes and eventually vanishes with sufficiently long running tasks and/or few servers.

4. Sharing of bandwidth to staging point

In the model described above, the transmission time T'_t was assumed to be independent of the number of simultaneous connections between the staging point and the computational servers. This assumption, in effect, indicates the availability of separate and independent connections between the staging host and the computational servers. For most real situations, the staging server has only one communication link that is shared among all connections to the computational servers. This means that as more simultanous connections are made to the staging server, the time to transfer the staged data on each connection is affected. This causes the effective bandwidth BW_e between the staging server and the computational servers to be different from the nominal bandwidth BW of the connecting link. In what follows, we derive a formula for the upper bound of BW_e. We maintain the assumptions that the total number of available computational servers is large enough to keep the client from being idle between tasks.

Consider Fig.3, which shows task instantiating using a staging server. Let S be the size of the data file that is staged

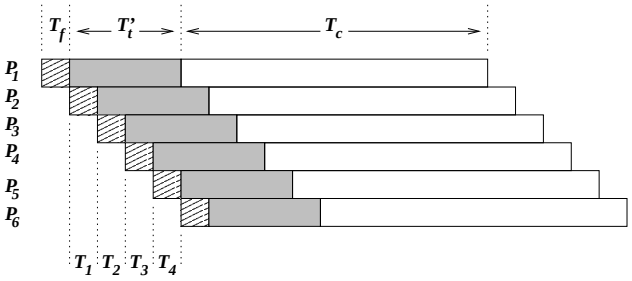

Figure 3. Sharing of bandwidth to staging server

and which is transmitted every time a new task is started. For simplicity, assume that the staged data file transmission time T'_t is given by $T'_t = mT_f$ for some integer m. Consider the first task started on P_1, during time interval T_1 (which is equal to T_f), task on P_1 has execlusive access to the communication link to the staging server. Hence, the amount of data transferred during T_1 is given by $S_1 = BW\ T_f$.

As a new task is started on P_2 , it shares the communication link to the staging host with the task on P_1 (assuming $S_1 < S$). Hence, the amount of data transferred to P_1 during time interval T_2 is given by $S_2 = (BW\ T_f)/2$

Similarly for tasks started on P_3 and P_4, each new task reduces the amount of bandwidth available to P_1. This process continues until time interval T_n during which data transmission for P_1 terminates. During time interval T_n, n simultaneous connections share the communication link to the staging server. During time interval T_{n+1}, task running on processor P_{n+1} starts transferring its instance of the staged data file, thus the communication link continues to be shared among n or more transmissions assuming the number of available computational servers is large enough. All subsequent transmissions will have an effective bandwidth $BW_e \leq BW/n$. The inequality stems from the fact that task started on P_2 will not necessarily finish transferring its input during time interval T_{n+1} (since it started with only half the nominal bandwidth). This allows yet more tasks to start, further reducing the bandwidth available to any one server. Then, we can write

$$\begin{aligned} S &= T_f\ BW + T_f \frac{BW}{2} + \cdots + T_f \frac{BW}{n} \\ &= T_f\ BW \sum_{i=1}^{n} \frac{1}{i} \end{aligned} \quad (9)$$

Then for $T'_t = T_f$, we have $S = T_f\ BW$ and $BW_e = BW$. For $T'_t = 2T_f$ we have $S = 3/2\ T_f\ BW$ and $BW_e \leq BW/2$. For larger values of T'_t, we can write

$$S \approx T_f\ BW[ln(n) + \gamma] \quad (10)$$

Complexity	T_f	T_t	T_t'	T_c	T_c'	P_e		P_e'		N		N'	
	(sec)	(sec)	(sec)	(sec)	(sec)	Calc.	Act.	Calc.	Act.	Calc.	Act.	Calc.	Act.
$n^{0.6}$	2.8	7.5	0.2	15.6	9.84	2.5	1.5	4.6	3.4	1050	1047	3895	3758
$n^{0.7}$	3.4	7.8	0.2	37.7	26.81	4.3	3.3	8.9	7.4	963	953	3168	2984
$n^{0.8}$	4.7	6.2	0.2	90.7	82.68	9.3	8.3	18.6	17.3	980	983	2280	2257
$n^{0.9}$	6.6	6.0	0.2	257.2	243.18	21.3	22.4	37.7	33.2	833	851	1478	1462
$n^{1.0}$	6.7	8.0	0.2	737.4	744.80	51.1	33.4	111.7	34.4	469	390	469	452
$n^{1.1}$	7.1	8.5	0.2	1458.8	1471.71	94.5	33.8	207.7	34.5	222	230	222	247

Table 1. Experimental results (1). P = 35, T = 3 hours

Complexity	T_f	T_t	T_t'	T_c	T_c'	P_e		P_e'		N		N'	
	(sec)	(sec)	(sec)	(sec)	(sec)	Calc.	Act.	Calc.	Act.	Calc.	Act.	Calc.	Act.
$n^{0.6}$	2.6	7.2	0.2	10.9	10.6	2.1	1.1	5.2	4.0	1104	1100	4198	4095
$n^{0.7}$	2.6	7.3	0.2	30.4	29.6	4.0	3.0	12.0	5.4	1075	1071	1989	1978
$n^{0.8}$	2.7	7.7	0.2	89.8	85.0	9.7	5.3	32.3	5.8	642	639	733	736
$n^{0.9}$	2.8	7.7	0.2	257.6	254.8	25.5	5.7	93.0	5.9	237	239	247	248
$n^{1.0}$	2.9	8.1	0.4	757.7	895.7	70.3	5.9	279.0	6.0	79	82	67	69
$n^{1.1}$	3.2	8.5	0.2	1515.8	1560.6	130.0	5.9	484.2	6.0	37	42	36	39

Table 2. Experimental results (2). P = 6, T = 3 hours

where γ is known as the *Euler-Mascheroni* constant and is approximately equal to 0.5772156649. It has been shown [18] that the error in the above approximation is bound by the formula

$$\frac{1}{2(n+1)} < \sum_{k=1}^{n} \frac{1}{k} - [ln(n) + \gamma] < \frac{1}{2n} \quad (11)$$

Note that the approximation in equation 10 can be used for all values of n, subject to the error constraints defined in equation 11. Solving equation 10 for n, we get

$$n \approx e^{\frac{S}{T_f \, BW} - \gamma} \quad (12)$$

equation 12 shows that the effective bandwidth connecting the computational servers to the staging point decreases exponentially as the staged data size increases relative to the maximum amount of data that can be transmitted from the staging server in time interval T_f. If n is not limited by the number of available servers, BW_e could decrease to the point where virtually no progress is made in the farming application as all instantiated tasks try to simultaneously retrieve their common input data set. We explore this effect further through simulations in section 6.

5. Experimental Results

In this section, we present the results of running a wide area distributed farming application. The goal of this part is to test the validity of the model developed in section 3 under real network and server loading conditions (computational servers running in work stealing mode). We developed a simple synthetic algorithm that allows us to control the running time of computational tasks through input parameters. Multiple instances of this algorithm were spawned on a network of computational servers from a remote client.

5.1. The testing environment

The experiments using the synthetic algorithm were performed using the following experimental setup:

- **Client:** The client is a SUN ULTRA-2 machine running at Princeton University.
- **Servers:** The computational servers are selected from a pool of 41 machines running at the University of Tennessee, Knoxville. Of these, 12 machines are dual 167-MHz UltraSPARC-1 processor machines with 256-MB of memory. The remaining 29 machines have single 143-MHz UltraSPARC processor with 256-MB of memory.
- **NetSolve agent:** The NetSolve agent was launched on one of the computational servers used in the experiment.
- **Staging Point:** The staging point is implemented by an IBP server [13] running on an Internet2 machine dedicated to serving storage to network applications [3]. This machine is housed at the University of Tennessee.

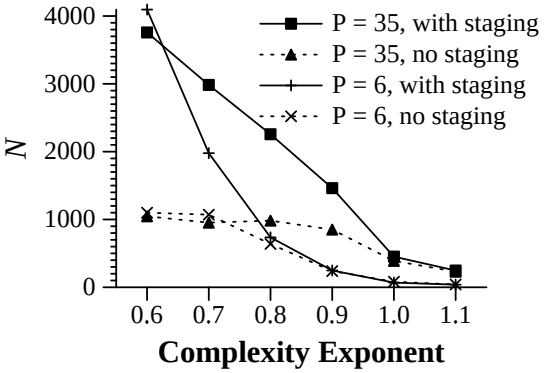

Figure 4. Completed tasks in each experiment.

- **Experimental setup:** All reported experiments were performed between the hours of 8:00 AM and 11:00 PM (EDT) during the months of May and June 2000. All components of the experiments were running in a work-stealing mode, with a *nice* value of 15.

The farming interface [5] in NetSolve was used to launch tasks and assign them to computational servers. However, this interface was instrumented to allow for better control over the maximum number of hosts that can be simultaneously used by a client, and to add various tracing and performance measurement code.

5.2. Results

In this section, we present results from two experiments. In both experiments, The duration of each task is set to be a function of the input data size n. These functions are of the form n^y, and in our tests, we vary y from 0.6 to 1.1. In the first experiment, the number of computational servers (P) is fixed at 35 In the second experiment, P is six. The goal of these experiments is to test the validity of the models developed in section 3 as well as the simulator code described later. All runs were conducted for a period T of 3 hours. Input data size for these experiment is 832 KB. Tables 1 and 2 summarize the results of these experiments.

As the last two columns of each table indicate, the model does a very good task of predicting actual behavior both with and without staging. And as is demonstrated most clearly in Table 1, the most notable improvement due to staging is the increase in P_e — roughly a factor of two in Table 1, and over a factor of three in Table 2. However, when there are not enough processors for P_e processors to be kept busy, as in the last two rows of Table 1 and the last four rows of Table 2, the reduced transmission time due to the input being near the servers has little effect.

Below we make a few more comments from the experiments:

- **Forking time T_f:** The average forking time increases as the number of computational servers used increases. The increase stems from the need for the client to negotiate with the NetSolve agent and servers to select a suitable server for every new task. As a result, the average starting time tends to increase as the number of available servers increases due to the dynamic server selection process. In all staging experiments, T_t' was considerably less than T_f; thus no bandwidth sharing effects were observed in the experiments we conducted.

- **Transmission time T_t:** In NetSolve, one can only measure total task instantiation time at the client. Without staging, this is $T_f + T_t$. With staging, it is simply T_f, since the data is downloaded from IBP after task instantiation. For that reason, T_t is calculated to be the task instantiation time without staging ($T_f + T_t$), minus the task instantiation time with staging (T_f).

- **Execution Time T_c:** The execution times listed in Tables 1 and 2 represent the time between the end of the client's task instantiation and the time at which the client detects task completion. As a result, this time may exceed the actual execution time, because the client does not check for task completion while in the midst of forking a new task. As a result, execution time as seen at the client may exceed the actual execution time by ($T_f + T_t$) seconds without staging, and by T_f seconds with staging. This accounts for the differences in T_c and T_c' in the tables.

Finally, in Figure 4, we plot the number of completed tasks for each test, both with and without staging. When $P_e < P$, the reduced task forking time of the $P = 6$ tests show better performance. Additionally, when $P_e < P$, staging again shows benefits because it increases P_e. When P_e reaches P, the benefits of staging are far less dramatic.

6. Simulation Results

In this section we report further results that we obtained through simulation. The simulator accepts the following parameters:

- Probability distributions for T_f, T_c, remote bandwidth from the client to servers, and local bandwidth from servers to their staging points.

- The input data size S, the number of servers P, and the application duration T.

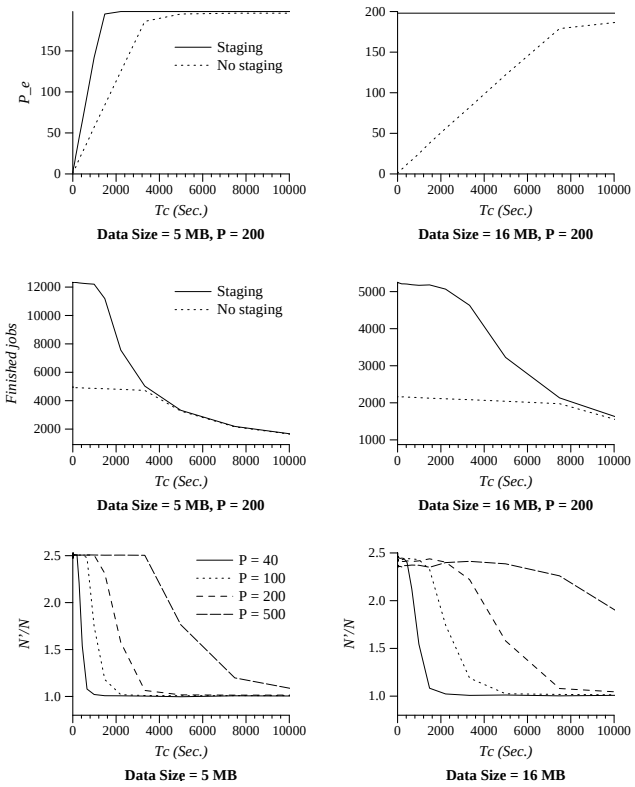

Figure 5. Simulation results, $T_f = [3, 7]$ Sec.

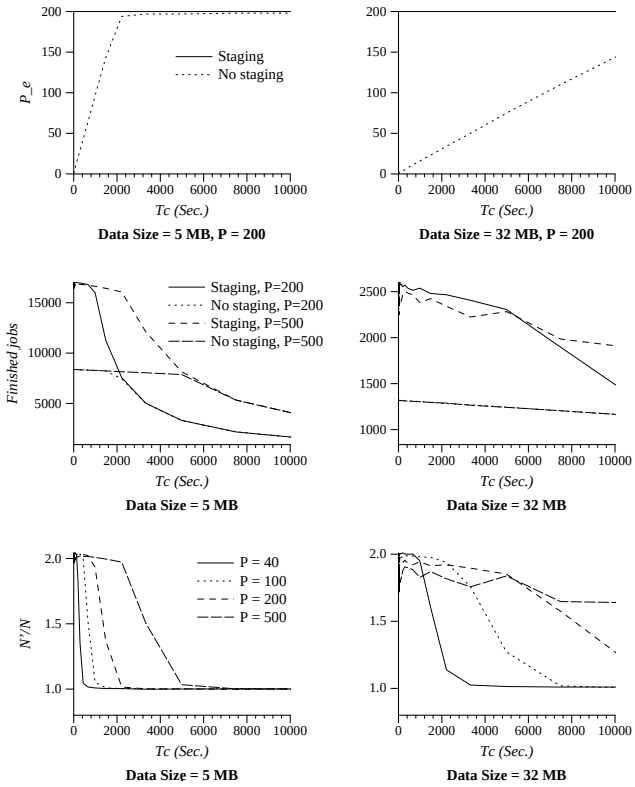

Figure 6. Simulation results, $T_f = [.05, .15]$ Sec.

It then performs a stochastic simulation of the farming application, and returns the number of tasks completed. The purpose these simulation tests is to explore the behavior of staging with respect to farming while varying the behavior of parameters For the runs described here, we sample T_c from a uniform distribution in the interval $[\mu - 0.25\ \mu, \mu + 0.25\mu]$, where μ is the mean value of T_c, and varied in these tests. Remote bandwidth is uniformly distributed in the interval $[0.25, 0.75]$ MB per second. Local bandwidth is uniformly distributed in the interval $[0.5, 1.5]$ MB per second. We vary the input data size S between 5 MB and 32 MB and the number of servers P between 40 and 500. Application duration T is fixed at 24 hours.

We used two settings for T_f to simulate two different scenarios. In the first set of results shown in Fig. 5, we use T_f uniformly distributed in the interval $[3, 7]$. This setting simulates remote dynamic scheduling where significant overhead is used in selecting computational servers. In the second set of results shown in Fig. 6, we use T_f uniformly distributed in the interval $[0.05, 0.15]$. This simulates a more static sheduling approach where computational servers are slected in a more time-eeficient manner (e.g. using round-robin scheduling).

Results shown in Fig 5 and 6 are the averages obtained over ten independent runs of the simulator. The figures show the results for the throughput ratio (N'/N),the number of finished tasks (N', N), and the effective number of computational servers used (P'_e, P_e). The values chosen represnt an environment with decent remote bandwidth (0.5 MByte/Sec) and local bandwidth that is twice that amount. The results reported here will be necessarily different for other bandwidth values, but the model and general conclusions should remain valid.

In Figure 5, the results are shown for input data sizes of 5 MB and 16 MB, and $P = 200$. In these graphs, three different zones can be identified in terms of the individual task computational time T_c:

Zone A This zone extends from $T_c = 0$ to approximately $T_c = 1200$ seconds. In this zone, neither model is able to utilize all available computational servers. The staging architecture is able to utilize more servers due to the reduced total task starting time.

Zone B This zone covers the interval $[1200, 3200]$ for a data size of 5 MB, and $[1200, 8000]$ for a data size of 16 MB. In this zone, the staging architecture is making full use of all available computational servers, while its no-staging counterpart is making partial use of them. It should be noted that the range of T_c for which this condition is true increases as the data size increases.

Zone C This zone covers the interval $T_c \geq 3200$ for a data size of 5 MB, and $T_c \geq 8000$ for a data size of 16 MB. For values of T_c in this zone, both architectures are making full use of all available servers. For sufficiently large running interval T, no significant difference is observed between the total number of finished tasks in both cases. For this region, staging does not improve task throughput.

It can be seen that the precise boundaries between the three zones depend on the particular configuration of the problem solving environment (various bandwidth values and number of available servers) as well as properties of the problem itself, namely input data size. For instances in zones A and B, staging is beneficial in improving task throughput, while for zone C, such improvement is not observed.

In Figure 6, simulation results are shown for a data size of 5 and 32 MB using 200 and 500 computational servers. In this scenario, the effects of bandwidth sharing can be observed for large input data sizes and small values of T_f. It can be seen that for values of T_c less than 5000 seconds, using 500 servers is actually yielding worse performance than the use of 200 computational servers. The increase in transfer time due to the increased number of simultaneous connections increases total task turnaround time to the point that nullifies the benefits gained from the extra computational servers. As T_c increases beyond 5000 seconds, the percentage of time spent transferring data is reduced to the point that we start seeing gains from the use of the extra 300 servers (although the gains are modest in the shown results).

7. Conclusion

In this paper, we have studied the performance of wide area data independent farming applications. We develop a model for total task throughput with and without staging. The staging model accounts for sharing of bandwidth to the staging point among the computational servers. The models are validated using experiments and simulations. The models and simulation results suggest that any improvement in task throughput through staging is primary due to increased server utilization through reduction in individual task starting time through staging. The effect of bandwidth sharing among the computational servers is most profound for large values of the ratio $S/(T_f\ BW)$, which could affect the decision to utilize more servers in the farming application.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parametised simulations using distributed workstations. In *The 4th IEEE Symposium on High Performance Distributed Computing*, August 1995.

[2] J. Basney, R. Raman, and M. Livny. High throughput Monte Carlo. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.

[3] M. Beck and T. Moore. The Internet2 Distributed Storage Infrastructure project: An architecture for internet content channels. *Computer Networking and ISDN Systems*, 30(22-23):2141–2148, 1998.

[4] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. *The Int. Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[5] H. Casanova, M. Kim, J. S. Plank, and J. Dongarra. Adaptive scheduling for task farming with grid middleware. *Int. Jrnl. of High Perf. Comp.*, 13(3):231–240, Fall 1999.

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *SC00 Conference on High-Performance Computing*, November 2000.

[7] M. Ferris, M. Mesnier, and J. More. NEOS and Condor: Solving optimization problems over the internet. Technical Report ANL/MCS-P708-0398, Argonne National Laboratory, March 1998. `http://www-fp.mcs.anl.gov/otc/Guide/TechReports/index.html`.

[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[9] I. Foster and C. Kesselman, editors. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, November 1998.

[10] A. S. Grimshaw, W. A. Wulf, and The Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[11] M. J. Lewis and A. Grimshaw. The core Legion object model. In *Fifth IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[12] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi. Utilizing the metaserver architecture in the Ninf global computing system. In *High Perf. Comp. and Network. '98, LNCS 1401*, pages 607–616, 1998.

[13] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *NetStore '99: Network Storage Symposium*. Internet2, `http://dsi.internet2.edu/netstore99`, October 1999.

[14] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf: Network based information library for globally high performance computing. In *Proc. Parallel Object-Oriented Methods and Appl.*, pages 39–48, Santa Fe, 1996.

[15] L. M. Silva, B. Veer, and J. G. Silva. How to get a fault-tolerant farm. In *World Transputer Congress*, pages 923–938, Aachen, Germany, September 1993.

[16] T. Tannenbaum and M. Litzkow. The Condor distributed processing system. *Dr. Dobb's Journal*, #227:40–48, February 1995.

[17] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Running EveryWare on the computational grid. In *SC99 Conference on High-performance Computing*, November 1999.

[18] R. M. Young. Euler's constant. *Math Gazette*, 75(472):187–190, 1991.