Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications

Atsuko Takefusa
Ochanomizu University
takefusa@is.ocha.ac.jp

Osamu Tatebe
Grid Technology Research Center, AIST
o.tatebe@aist.go.jp

Satoshi Matsuoka
Tokyo Institute of Technology / National Institute of Informatics
matsu@is.titech.ac.jp

Youhei Morita
High Energy Accelerator Research Organization (KEK)
youhei.morita@kek.jp

Abstract

Data Grid is a Grid environment for ubiquitous access and analysis of large-scale data. Because Data Grid is in the early stages of development, the performance of its petabyte-scale models in a realistic data processing setting has not been well investigated. By enhancing our Bricks Grid simulator to accomodated Data Grid scenarios, we investigate and compare the performance of different Data Grid models. These are categorized mainly as either central or tier models; they employ various scheduling and replication strategies under realistic assumptions of job processing for CERN LHC experiments on the Grid Datafarm system. Our results show that the central model is efficient but that the tier model, with its greater resources and its speculative class of background replication policies, are quite effective and achieve higher performance, while each tier is smaller than the central model.

1. Introduction

Data Grid is a Grid environment for ubiquitous access and analysis of large-scale data. Several recent projects have been designed and implemented to analyze data that will be derived from scientific experiments such as the CERN LHC (Large Hadron Collider) experiment [15] starting in 2007. Grid technology will play an essential role in constructing worldwide data-analysis environments where thousands of physicists will collaborate and compete in particle physics data analysis on new energy frontiers. To process such large amounts of data, a global-scale Grid computing model consisting of multi-tier worldwide Regional Centres has been studied by the MONARC project [2].

Because petascale data is seldom updated and is often accessed in read-only mode. This means that file replication can be used to promote the efficient sharing of such large-scale data worldwide by improving load balancing, access bandwidth, disk usage, and fault tolerance. The basic technologies needed for file replication are fast long-distance file transfer and efficient worldwide replica management. Moreover, efficient execution depends on the management of scheduling issues, such as how to select the best replication procedure, how to choose the most appropriate compute nodes, and how to allocate output and temporary file space. File replication adds the complexity of data movement to the scheduling issues: for instance, what file is to be replicated, when and where the replication is to occur, and also when and which replica is to be deleted to maintain storage capacity.

However, since Data Grid is still in the development stage, there has not been sufficient investigation into the suitability of proposed Data Grid architectures or into the performance of Data Grid systems in large-scale, realistic applications with various scheduling and replication policies.

We simulate the performance of different Data Grid system models that employ various replication and scheduling strategies. These simulations apply realistic assumptions of job processing for CERN LHC experiments to the Grid Datafarm system [4, 14] by using the Bricks Grid simulator [3]. This simulator is enhanced with new Data Grid operation extensions, mainly for comparing centralized data storage and processing vs. MONARC-style hierarchical distributed configuration. Our results show that the central model is efficient

but that the tier model, with its greater amount of resources and its speculative class of scheduling and replication policies, achieves higher performance, while each tier is smaller than the central model.

2. Grid Datafarm Architecture

For efficient data processing of data-intensive applications, both data access bandwidth and CPU power must increase as the amount of data increases. In order for the Data Grid to offer shared petabyte-scale data processing, it will be necessary to have not only fast file transfer and efficient replica management but also fast data access and processing. Even TB/s-scale bandwidth is not enough to process petabyte-scale data. In general, such bandwidth scaling to TB/s does not seem to be feasible in a distributed computing environment such as the Grid. However, large-scale data-intensive computing frequently involves high degrees of local data access, which could be exploited to compensate for the lack of global bandwidth.

Because of the localized nature of such data access, and because of salient Grid properties such as scheduling, load balancing, fault tolerance, security, etc., it would be disadvantageous to have a separate I/O across the network independent from the compute nodes. It would be better to strive for tight coupling of storage to the computation to achieve terascale data processing goals. In other words, it is better to adopt the "owner computes" strategy, or the "move the computation to the data" approach, than to adopt the "move the computation to the data" approach used by most data-intensive processing systems, such as HPSS [6]. For highly data-parallel applications, the owner-computes strategy is much more scalable and is far better suited to the requirements of the Grid.

To fully exploit the localization of data access for scalable I/O bandwidth, local I/O bandwidth of the local disks on each node should be utilized. In the Grid Datafarm architecture, every node has large and fast local disks, and acts as both a storage node and a compute node, which together are called a *Gfarm filesystem node*. When each node has 1-TB of local storage, a 1000-node cluster provides an online 1-PB storage, which is managed by a *Gfarm filesystem* as a single filesystem image in a scalable fashion.

The main features of the Grid Datafarm are to support extreme disk I/O bandwidth scales to the TB/s range and to support file replica management for fault tolerance and load balancing. To achieve an extreme disk I/O bandwidth, the local disk I/O of each node is naturally exploited by two new features: file-affinity scheduling and a local file view. Any file can be

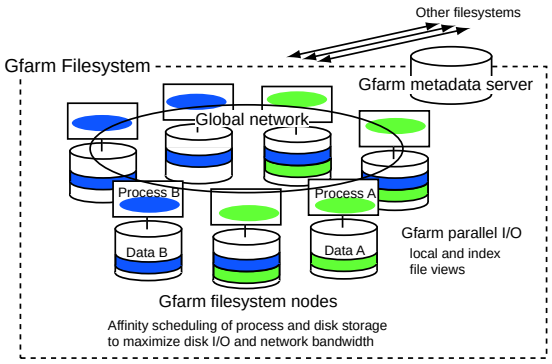

Figure 1. Gfarm file system.

replicated anytime for fault tolerance and to improve system performance, which is managed by filesystem metadata consistently. The replica catalog will be utilized by a scheduler or a resource broker to determine the best storage-and-compute nodes.

The Gfarm filesystem is a parallel filesystem for petascale data-intensive computing on a Grid using local disks of every node. Figure 1 depicts the components of the Gfarm filesystem, *Gfarm filesystem nodes* and *Gfarm metadata servers*, that provide huge, petabyte-range disk space with scalable disk I/O bandwidth and fault tolerance.

Each Gfarm filesystem node has large and fast local disks, so it acts as both a storage node and a compute node. Running on each filesystem node is an I/O daemon, called *gfsd*, to facilitate remote file operations with access control as well as user authentication, file replication, fast program invocation, and node resource status monitoring. A metadata server manages filesystem metadata including owner, access permission, access times, and replica catalog data.

The filesystem manages a ranked group of files as a single Gfarm file. Each file in a Gfarm file is called a *Gfarm file fragment* and is stored on a filesystem node or on several such nodes by file replication. By default, a file group can be accessed as a single large file in its *global view*, or its constituent file fragments on each node can be accessed individually in either *local* or *index file* view. This is facilitated by the Gfarm parallel I/O APIs. The filesystem could be regarded as an extension of a striping cluster filesystem in that each file fragment has an arbitrary length and can be stored on any node. Refer to [14] for further detail, including other features such as fast file transfer and replication, file recovery and regeneration, and performance evaluation.

In this evaluation, we assume the Grid Datafarm as

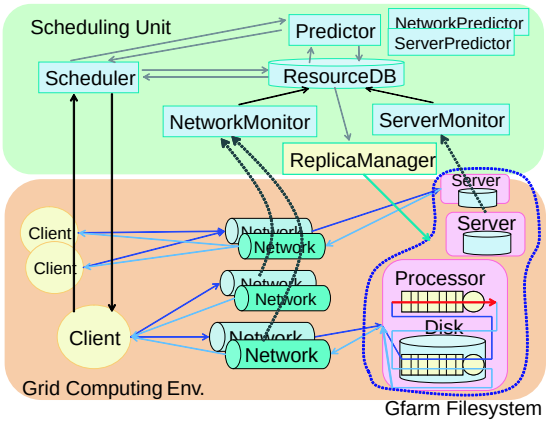

Figure 2. The Bricks architecture.

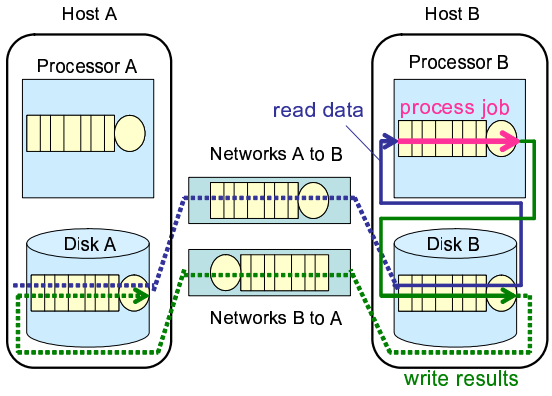

Figure 3. Bricks simulation steps for Data intensive computing.

theunderlying Data Grid architecture being simulated, but our results are applicable to other Data Grid systems in that the only assumption we make on GFarm is its massive local bandwidth and that the owner-computes rule is the norm unless replication occurs. A large Data Grid systems is necessary in either case.

3. Data Grid Extension of the Bricks Grid Simulator

The Bricks Grid simulation framework [3] is a Java-based discrete event simulator that allows users to evaluate the performance of various scheduling algorithms and scheduling framework components in Grid environments. Bricks provides canonical Grid scheduling modules (called the Scheduling Unit) and various scheduling-analysis-simulating dynamic Grid environments (called the Grid Computing Environment) [13, 12] (Figure 2).

In order to evaluate the performance of various Data Grid application scenarios, we extended Bricks to simulate the following features:

- Local disk I/O overheads
- Replica Manager
- Replica Catalog
- Disk management mechanism

3.1. Local Disk I/O Overheads

For accurate simulation of petabyte-scale data intensive computing, local I/O access overhead becomes non-negligible. We therefore extend Bricks to represent the behaviors of storage, such as local I/O overhead and data management, with additional (storage) queues. Each job in the Bricks simulation is processed as shown in Figure 3. The solid lines indicate a job workflow for an owner-computes case: read data from Disk B, process the job on Processor B, and store the results on Disk B. Conversely, the solid+dotted lines indicate a job workflow when data required for a job is not stored locally on the compute processor: read data from Disk A via Networks A to B and Disk B, process the job on Processor B, and store the results on Disk A via Disk B and Networks B to A. (There could be other situations, such as when the data is remote but the results are stored locally.) In the simulation, Processor and Disk queues process jobs in a time-sharing manner and Network queues process them in a first-comes-first-served manner with logical packet transmissions. Each Disk queue is shared by both data read and data write.

3.2. Replica Manager and Replica Catalog

For efficient data processing, the distribution of data required by each data-intensive job is important not only for the scalability of data access bandwidth and CPU power, but also for load balancing and fault tolerance, as described in Section 2. Two new modules, a replica manager and a replica catalog, are added in the Scheduling Unit. The replica manager takes care of the background file replication algorithms described in Section 5.2. It periodically collects monitoring information over Grid resources and controls file replica creation, migration, and even elimination, all in the background. The replica catalog module manages a

list of mappings from a logical filename to a physical storage node to support a Gfarm filesystem.

3.3. Disk Management Mechanism

"Disk management" is an important issue for Data Grid systems because petabyte-scale data for data-intensive applications cannot be stored on any local disk on the Grid, and because replica generation for load balancing and fault tolerance will further reduce available disk space. In order to avoid a shortage of disk space, Bricks manages each amount of available disk space on the Grid and eliminates data appropriately. The details of our replica elimination algorithm are described in Section 5.3

4. Simulation Modeling

4.1. Modeling of Large Data Grid Application: CERN LHC Experiments

In LHC experiments at CERN, observed data (events) are collected from a huge number of collisions of particles and analyzed through different levels of a data processing hierarchy [2] as follows:

Large - RAW → ESD : Reconstruct RAW observed data, create ESD (Event Summary Data object) (2-4 times/year).

Medium - ESD → AOD : Using ESD, redefine AOD (Analysis Object Data) (once/month).

Small - AOD → TAG : Using AOD, redefine TAG data (once/4 hours).

A typical job for each job class (Large, Medium, or Small) in the experiments is an independent parallel data processing of a collection of millions of physical events. Each job is handled on a Data Grid system as follows:

1. A user (a physicist) at the client machine invokes a job.
2. The Data Grid scheduler selects a suitable set of servers.
3. Each server loads a data fragment required for the job, over the Grid if the fragment is non-local.
4. Each server processes the individual portions of the job that have been assigned to it.
5. The servers send the output to specified storages locations (the client receives only statistical data).

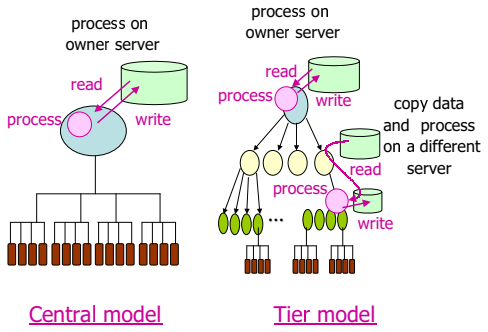

Figure 4. Central model (left) and Tier model (right).

The processing time for a job is given as:

$$ResponseTime = Read + Process + Write \quad (1)$$

$ResponseTime$, $Read$, $Process$, and $Write$ indicate, respectively, the total duration of job processing on the Data Grid and the time it takes to read the input data, process the job, and write the output data.

4.2. Data Grid Architecture

The MONARC project proposed a multi-tier regional center model. At the time the MONARC report was published, the model was assumed to be necessary due to limitations on the computational and storage resources that could be concentrated at a single site. However, remarkable improvement in commodity PC technologies, as well as advances in clustering technologies, may make it feasible to construct huge clusters with petascale online storage, thus providing appropriate data processing capacity, as reported in [2]. In fact, in the Grid Datafarm project we constructed several design studies and proof-of-concept large-scale storage cluster prototypes that increasingly convinced us that such clusters could be constructed feasibly by 2007. Thus, a meaningful comparison would be to investigate how much of a performance penalty we might suffer by distributing storage and computational resources in the MONARC style.

Towards this end, and as observed in Figure 4, we compare two categories of models: the central model, in where all the jobs are processed at a single site, and the MONARC-style tier model, in which jobs are processed according to their hierarchical levels. For

the tier model, Data Grid systems must facilitate suitable scheduling and replication policies to deploy user jobs and maintain data replicas over the resources for efficient data processing. We discuss the details of scheduling and replication algorithms for tier models in Section 5.

5. Scheduling and Replication Policies for Tier Model

Tier-based Data Grid systems must facilitate suitable scheduling and replication. On the one hand, creating a large number of replicas for this purpose balances the load and minimizes response time. On the other hand, storage capacity and network bandwidth pressures become significant, penalizing performance. To resolve this situation, we evaluate several on-line scheduling and replication policies, some of which we originated, and combine them in valid ways (note that some combinations do not make sense) under actual HEP application scenarios.

5.1. On-line Scheduling Algorithms and Policies

To optimize performance, the on-line scheduler determines the DataSourceHost (owner of input data), ComputeHost (computational resources), and DataDestinationHost (store result values) for each incoming request, given the current status of the system. If DataSourceHost != ComputeHost or ComputeHost != DataDestinationHost, then input or output data are replicated "on demand". On the other hand, the Replica Managers periodically collect monitoring information over the Grid resources and manage replica creation, migration, and their elimination "in the background".

The set of on-line scheduling algorithms is as follows:

Greedy : The greedy on-line scheduling algorithm described in [7] as MCT (Minimum Completion Time). The scheduler assigns the request to whichever host completes it firstest in Equation (1). The job reads the input data from the appropriate host, and the output is stored on the compute host. All subsequent scheduling policies are based on MCT.

OwnerComputes : The scheduler selects whichever compute host owns the input data and completes the request first. In this policy, DataSourceHost, ComputeHost, and DataDestinationHost are dispatched to the same host.

LoadBound-Read : The scheduler selects a compute host with MCT from the host group that satisfies:

$$Perf_{Specified} > Perf_{Estimated} \quad (2)$$

$$Perf_{Estimated} = ProcPerf/(LoadAvg + 1) \quad (3)$$

$ProcPerf$, $LoadAvg$, $Perf_{Estimated}$, and $Perf_{Specified}$ indicate the performance of a compute host, monitored load average of the host, estimated performance of the host, and a scheduling parameter to determine the frequency of replications, respectively. Specifying a smaller $Perf_{Specified}$ will increase the number of on-demand replications. The job reads input data from the appropriate host, and the output is stored in the ComputeHost.

LoadBound-Write : The scheduler selects whichever compute host has the smallest $ResponseTime_{Estimated}$:

$$ResponseTime_{Estimated} = Read_{Estimated} + Process_{Estimated} \quad (4)$$

If the selected host does not satisfy Equation (2), then the result is sent to the host that maximizes Equation (3).

5.2. Replication Policies

Replica Managers periodically collect the status of each host's resources and trigger replica creation and migration in the following way:

LoadBound-Replication : The Replica Manager periodically computes $Perf_{Estimated}$ in Equation (3) for all the hosts. If Equation (2) is satisfied, the Replica Manager triggers replication from the host with the smallest $Perf_{Estimated}$ to the host with the largest $Perf_{Estimated}$.

The Replica Manager actually creates a replica for data with the largest Access Rate AR:

$$AR = N_{Accesses}/(T_{Current} - T_{Stored}) \quad (5)$$

$T_{Current}$ and T_{Stored} indicate the current time and the time the data was stored, while $N_{Accesses}$ indicates the total number of times the data was accessed in the period from T_{Stored} to $T_{Current}$.

Aggressive-Replication : Aggressive-Replication is the background replication algorithm by which the

Replica Manager always generates a replica of all the data generated by each job. When a client host notifies the Replica Manager that a job has been processed, the Replica Manager generates a replica of the data and sends it to whichever host maximizes $Perf_{Estimated}$.

Our simulations employ the combinations of four scheduling algorithms (OwnerComputes, Greedy, LoadBound-Read, and LoadBound-Write) and three replication policies (LoadBound-Replication, Aggressive-Replication, and "Do Nothing" (no background replication policies)).

5.3. Replica Elimination Algorithm

We assume that some host will be the "home" for the original archival storage of any given data, and that all data transmission is for data replication, i.e., data are copied rather than moved. For example, if the DataSourceHost and the ComputeHost are different, all data required for the job is copied from the DataSourceHost storage to the ComputeHost storage. If the storage space for a job turns out to be insufficient (checked by the Scheduler), or some x% of hosts do not embody some y% of available storage space within the entire Data Grid, "*replica elimination*" is performed (the parameters x and y are specified at run time). We use the following "replica elimination" algorithm for over-capacity storage:

1. Select data that have replicas within the Data Grid environment.
2. Sort all the selected data by the last recently used (LRU) time.
3. Calculate $AR_{Elimination}$ for the first N data on the list:

$$AR_{Elimination} = AR/N_{Copies} \quad (6)$$

   N_{Copies} indicates the total number of replicas within the environment.

4. Select and eliminate replica for data with minimum $AR_{Elimination}$ from the first N data while maintaining the following condition:

$$TotalDiskSize \times Compactness > AvailableDiskSize \quad (7)$$

   $TotalDiskSize$ and $AvailableDiskSize$ indicate the total and available storage space on the entire Data Grid, while $Compactness$ shows the parameter to control the frequency of replica eliminations.

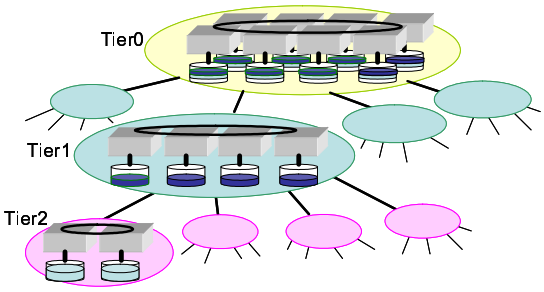

Figure 5. Simulated Grid Datafarm architecture.

5. If Equation (7) is not satisfied, return to Step 3 for the next N data.

In our simulation, N is set to 10.

6. Experiments

We investigate the performance of the central and tier models under various scheduling and replication policies. We compare their job response times on the Grid Datafarm architecture.

6.1. Experimental Environment for the Grid Datafarm Architecture

In our experiments, we assume a Data Grid model (Figure 5) based on central or multi-tier worldwide regional centers as considered in [2]. Each regional center is assumed to be based on the Grid Datafarm architecture described earlier. Each node in each regional center has local local disks (some large and some fast ones) that are mainly utilized for data processing. On the Grid Datafarm architecture, the aggregation of local I/O bandwidth to process a job is as follows:

$$TotalI/OBandwidth = LocalI/OBandwidth \times \#ofnodes \quad (8)$$

$\#ofnodes$ indicates the number of nodes on the site.

6.2. Simulation Scenarios

We compare the two models described in Section 4.2.

Central : All data is stored and processed at the central site, where we assume has sufficient processing power to handle all jobs. According to the queuing theory, all the relevant queues of jobs at the site should be stable.

Table 1. Parameters set for simulated Data Grid environments.

Model	Storage [PB]	Performance [MSI95]	# of Nodes in the Site	Total I/O Bandwidth
Central	2	0.5-1.8	10000	1 [TB/sec]
Tier	Tier0(x1): 2	0.6/0.5/0.4	10000	1 [TB/sec]
	Tier1(x4): 1	0.3/0.25/0.2	5000	500 [GB/sec]
	Tier2(x16): 0.1	0.03/0.025/0.02	500	50 [GB/sec]

Table 2. Parameters for HEP jobs. The number of events for a job is 1G.

Job	Comp. Size [GSI95*sec]	Frequency	Input [TB]	Output [TB]
Large	1000	1/4 [months]	1000	100
Medium	25	1/1 [month]	100	10
Small	5	1/4 [hours]	10	0.1

Tier : When the server load increases, a copy of the data is created at a lower-level tier and job processing is delegated to that tier. In this model, we use the 12 variations of scheduling and replication policies described in Section 5.

Table 1 shows the set of parameters for the experimental simulation environment, which is similar in its settings to the simulations performed in [9]. For the Tier model, we assume three different settings: (Tier 0, Tier 1, Tier 2) = (0.6, 0.3, 0.03), (0.5, 0.25, 0.025), (0.4, 0.2, 0.02) [MSI95(10^6SpecINT95)]. The queuing theory indicates that the Central model with 0.453318 [MSI95] becomes saturated and cannot process LHC jobs. WAN bandwidth and local I/O bandwidth are set to 10 [Gbps] and 100 [MB/sec], respectively, assuming a viable technology level in 2007. Data access for every job is highly localized because each job processes 10^9 events independently. Using a local file view and file-affinity scheduling, scalable local I/O bandwidth can be exploited to access large-scale data in a single system image. Assuming each local I/O bandwidth is 100 MB/s, the data access bandwidth at each site can be scaled up to the bandwidth listed in the 'Total Disk I/O' column in Table 1.

We have analyzed jobs running different levels of analysis in actual LHC experiments (Section 4.1) and typified them into three concrete instances, as shown in Table 2. In the LHC experiments, the amount of data for RAW, ESD, AOD, and TAG is expected to increase as shown in Table 3. The numbers in parentheses indicate the number of "original" data excluding replicas across the environment.

We then conduct thousands of one-year simulations, which start with Phase 3 and finish at the end of Phase 6, using the Presto III cluster (Dual Athlon MP 1900+, 768MB memory, 256 nodes) at the Tokyo Institute of Technology. Initially all data (1000TBx1, 100TBx2, 10TBx4) resides at Tier0 in all the scenarios. We assume that each 1000TB RAW data takes shelter in a different storage space, such as HPSS, and that the storage space for the data does not increase during a simulation.

Of the data access patterns characterized in [9], our simulation assumes both random patterns (no correlation) and temporal localization patterns (recently accessed files are likely to be accessed again). Other types of locality, such as geographical locality and spatial locality, are not employed, as it is not obvious whether other localities will be significant in LHC experiments.

6.3. Experimental Results

We compare the response time for the Central model against that for the Tier model as shown in Figures 6, 7, 8, and 9. These are the results after ten repetitions of the simulation for each combination of model and scheduling/replication algorithms in the temporal locality or random access patterns. The total numbers of jobs for Large, Medium, and Small were 30, 102, and 21693, respectively.

Figure 6 shows the response time for the Central model. The performance of Tier 0 varies from 0.5 to 1.8 [MSI95]. The 'large', 'medium', and 'small' correspond to job class Large, Medium, and Small, while 'to-

Table 3. The average increase of RAW(1PB), ESD(100TB), AOD(10TB), and TAG(10GB).

Phase	Data (#)
0mth	1PB(1)
4mth	1PB(2), 100TB(1)
8mth	1PB(3), 100TB(2), 10TB(4)
12mth	1PB(4), 100TB(3), 10TB(8), 10GB(720)
16mth	1PB(5), 100TB(4), 10TB(12), 10GB(1440)
20mth	1PB(6), 100TB(5), 10TB(16), 10GB(2160)

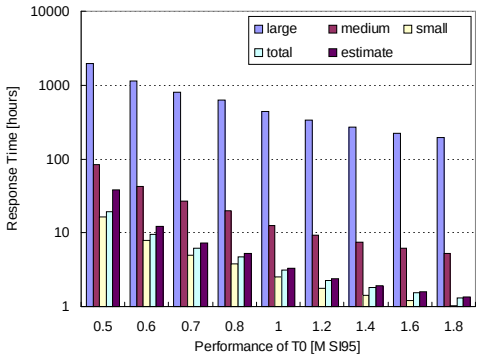

Figure 6. Central **model response time. Tier0 performance varying from 0.5 to 1.8 [MSI95].**

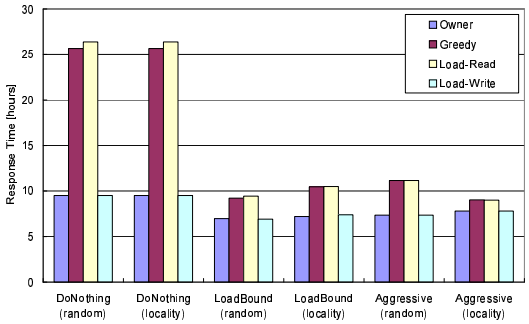

Figure 7. Tier **model response time with various scheduling and replication policies and temporal/random access pattern. (Tier0, Tier1, Tier2) = (0.6, 0.3, 0.03) [MSI95].**

tal' and 'estimate' indicate the average response time of all jobs and the estimated computation duration calculated using the queuing theory. The x-axis indicates the performance of a central site in MSI95 and the y-axis indicates response time in log scale. Figure 6 shows that response time drastically increases as the performance of the central site decreases.

Figures 7 and 8 show the average response times for the Tier model with the 12 different combinations of scheduling and replication policies. In this LHC scenario, AODs generated by the Medium jobs are surely accessed by lots of Small jobs that are frequently in the proximal future. We specified the Scheduler and Replica Manager to generate replicas of AODs during the simulations. The performances of each tier are 0.6, 0.3, 0.03 or 0.5, 0.25, 0.025 [MSI95]. The x-axis shows the three replication policies (DoNothing, LoadBound-Replication, and Aggressive-Replication), data access patterns, and both random and temporal localization. DoNothing means that no background replication policies are used. 'Owner', 'Greedy', 'Load-Read', and 'Load-Write' indicate the four scheduling policies (OwnerComputes, Greedy, LoadBound-Read, and LoadBound-Write, respectively).

For all cases, Greedy and LoadBound-Read show lower performance than even the OwnerComputes algorithm without file replication. That is because the overhead of on-demand file replication of hundreds of terabyte data before processing is too huge to overcome. On the other hand, all the scheduling policies with two of the replication policies, LoadBound-Replication and Aggressive-Replication, exhibit drastic performance improvements in average response time compared to cases without file replication (, i.e., in DoNothing).

Figure 9 compares response time in hours among different settings of the Central and Tier models. 1.2 [MSI95] equals the performance of 10,000 of 2.8 GHz Pentium 4 processors. Although response time with the Central model improves with higher Tier 0 site performance, the system quickly becomes unstable in resource-starved situations. This is a cause for concern, since there could be stringent limitations on the resources that could be placed on one site.

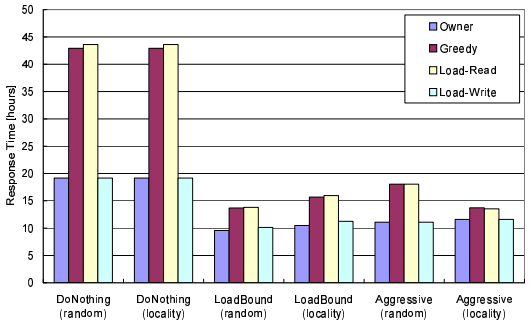

Figure 8. Tier **model response time with various scheduling and replication policies and temporal/random access pattern. (Tier0, Tier1, Tier2) = (0.5, 0.25, 0.025) [MSI95].**

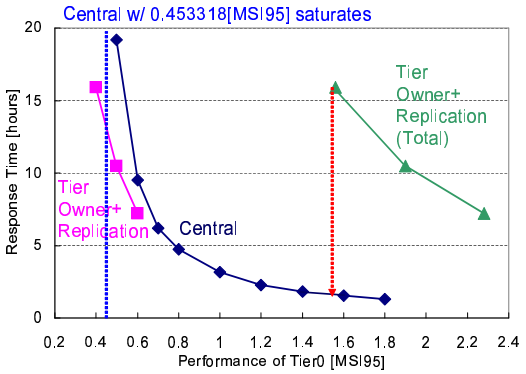

Figure 9. Comparison of response time between Central **model and** Tier **model with** OwnerComputes **and** LoadBound-Replication**. In** Central**, the minimum performance to maintain stability is 0.453318 [MSI95] according to the queuing theory.**

Given more resources, we could sustain system stability and achieve higher performance even when each tier is smaller than the Central model. In particular, we observe that even if Tier 0 is smaller than the stability threshold, we achieve stability and performance comparable to that of a larger Tier 0 setting. In such a case, employing a speculative class of scheduling and replication policies proves to be effective (such as with LoadBound-Replication). On the other hand, considering the aggregate performance of all sites that is shown by the triangle points in Figure 9, there is plenty of room for improvement in the performance.

7. Related Work

To study Grid framework components, scheduling algorithms, and application performance, reproducible and controlled evaluation is indispensable under various resource settings. This sort of evaluation, however, is too difficult and too costly to be obtained by a physical Grid environment. Typically, there are two approaches to achieve this goal.

The first approach is to emulate a Grid environment. The paper [11] describes a computational Grid emulator called MicroGrid. It virtualizes resources and emulates a virtual Grid environment, allowing various performance settings against which real Grid middleware, benchmark programs, and applications can be evaluated. Unfortunately, there is no support for virtualized storage resources at this time. If you have Grid schedulers and real applications, it is possible to evaluate different scheduling algorithms with various environment settings, although the emulation cost is prohibitive for very large numbers of large-scale applications in a worldwide Grid environment.

The second approach is to simulate a Grid environment using a discrete event simulator. Although there are several Grid simulation tools, we introduce the MONARC simulation tool [2, 1] and ChicSim [10] because both support the simulation of the storage needed for data-intensive computing.

The MONARC simulation tool is an object-oriented discrete event simulator written in Java. The tool employs a process-oriented approach for flexible simulations and consists of threaded objects or "Active Objects". The data model follows the Objectivity/DB architecture, which is a basic object data design used in HEP. The database server component simulates a client-server mechanism used to access objects from a database. However, to the best of the authors' knowledge, there have been no study thus far of scheduling and file replication algorithms using this simulation tool.

ChicSim (the Chicago Grid Simulator) is also a Grid simulator targeted for the CERN LHC experiment in the GriPhyN [5] project. It is built over Parsec [8], a C-based parallel simulation language. The study [10] on ChicSim showed the performance of various combinations of External Schedulers and Dataset Schedulers under simulative environments with smaller data sets. Although such small-grain benchmarking is quite important, their simulation was not conducted with the assumption of large application scalability. Our sim-

ulation attempts to simulate the scalability of actual application scenarios involving thousands of jobs of realistic sizes and their respective intervals.

8. Conclusions

We introduced several process scheduling algorithms, file replication algorithms, and a replica elimination algorithm on a Data Grid, and discussed the effectiveness of different combinations of scheduling and file replication algorithms under realistic Data Grid scenarios using the Grid Datafarm architecture and the Bricks Data Grid simulator. The results described in the previous section under a realistic LHC job model and a realistic worldwide multi-tier data Grid environment show the following.

- The average response time of the Central model improves with higher Tier 0 site performance, but the system quickly becomes unstable in resource-starved situations.
- With background file replication algorithms (LoadBound-Replication and Aggressive-Replication), , all of the scheduling algorithms, especially the OwnerComputes algorithms, exhibit drastic performance improvement in average response time compared with scheduling without background file replication.
- Even when a single site does not have enough resources, the OwnerComputes scheduling algorithm, combined with the LoadBound-Replication background replication, helps to avoid resource-starved situations.

Acknowledgments

This research was partially supported by the Ministry of Education, Culture, Sports, Science, and Technology (MEXT), Grant-in-Aid for Scientific Research on Priority Areas, 13224034.

References

[1] MONARC Simulation Tool. `http://monarc.web.cern.ch/MONARC/simulation/`.

[2] M. Aderholz and et al. Models of networked analysis at regional centres for lhc experiments. Monarc phase 2 report, 2000.

[3] Bricks. `http://ninf.is.titech.ac.jp/bricks/`.

[4] Grid Datafarm. `http://datafarm.apgrid.org/`.

[5] GriPhyN. `http://www.griphyn.org/`.

[6] HPSS: High Performance Storage System. `http://www.sdsc.edu/hpss/`.

[7] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[8] PARSEC. `http://pcl.cs.ucla.edu/projects/parsec/`.

[9] K. Ranganathan and I. Foster. Identifying Dynamic Replication Strategies for a High Performance Data Grid. In *Grid Computing*, 2001.

[10] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, pages 352–358, 2002.

[11] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: a Scientific Tool for Modeling Computational Grids. In *Proceedings of SC2000*, 2000.

[12] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In *Proc. of HPDC-10*, pages 406–415, 8 2001.

[13] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *Proc. of HPDC-8*, pages 97–104, August 1999.

[14] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi. Grid Datafarm Architecture for Petascale Data Intensive Computing. In *CCGrid2002*, pages 102–110, 2002.

[15] The Large Hadron Collider. `http://www.cern.ch/lhc/`.