Performance Evaluation of Data Management Layer by Data Sharing Patterns for Grid RPC Applications*

Yoshihiro Nakajima[2], Yoshiaki Aida[1], Mitsuhisa Sato[1], and Osamu Tatebe[1]

[1] Graduate School of Systems and Information Engineering,
University of Tsukuba, Tsukuba, Japan
{aida,msato,tatebe}@hpcs.cs.tsukuba.ac.jp
[2] NTT Network Innovation Laboratories,
Nippon Telegraph and Telephone Corporation, Tokyo, Japan
nakajima.yoshihiro@lab.ntt.co.jp

Abstract. Grid RPC applications, typically master-slave type of applications, often need to share large size of data among workers. For efficient and flexible data sharing among a master and workers, we have designed and developed a data management layer called OmniStorage. This paper enhances the OmniStorage functionality to accommodate several data transfer methods and to specify a hint for data sharing patterns, and develops a set of synthetic benchmarks based on data sharing patterns required by grid RPC applications to evaluate the performance and characteristics of each data transfer method. The performance evaluation and the hint help to select a suitable data transfer method, which improves the efficiency and also scalability of grid RPC applications that need to share large size of data among a master and workers.

1 Introduction

Grid technology enables integration of the computing resources in the wide-area network and sharing of huge amounts of data geographically distributed in several places. In order to make use of computing resources in a grid environment, an RPC-style system is particularly useful in that it provides an easy-to-use, intuitive programming interface that allows users of the grid system to easily develop grid-enabled applications. Several systems adopt Grid RPC as a basic model of computation, including Ninf [1], NetSolve [2] and DIET [3]. We have developed a grid RPC system called OmniRPC [4] for parallel programming solution in clusters and grid environments.

Grid RPC applications such as parametric search programs and task parallel programs, sometimes require a large amount of shared data among a master and workers. For instance, in some parametric search applications, the workers require a large common initial data and different parameters to execute different computations at remote nodes in parallel. In the RPC model, a master issues a remote procedure call and receives results from the invoked remote procedure. When the master needs to send the same and large initial data to every worker by arguments of the remote procedure call,

* This study was supported in part by MEXT KAKENHI (No. 17200002, 17·7324, 19024009), and by the Japan-France collaboration research program (SAKURA) through the JSPS. The study was performed in University of Tsukuba.

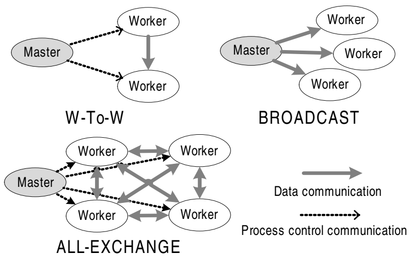

Fig. 1. Synthetic data sharing patterns

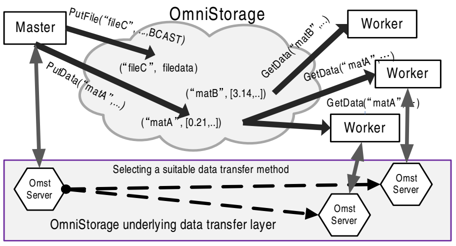

Fig. 2. Overview of OmniStorage

the data transfer from the master is wasteful and also causes a bottleneck for scalability. Thus, we proposed a programming model to decouple the data transmission from the RPC model to allow the data to be transferred efficiently by a data management layer [5]. The data management layer provides a temporal storage space that can be shared by both a master and workers, and APIs for data registration and data retrieval to the shared data.

For the data management layer of grid RPC applications, we have designed and developed OmniStorage [5]. OmniStorage provides APIs to access data using an identifier and hides both an internal behavior of data transfer method and a data placement from the application level. In [5], a preliminary design of OmniStorage using tree-topology-aware file broadcasting and the performance results are described. This paper enhances the OmniStorage functionality to exploit several data transfer methods in data transfer layer and to accept a hint for data sharing patterns in order to help to choose a suitable data transfer method.

Several studies on the data management layer gave some performance evaluation using simple benchmarks [6,7], but the lack of synthetic workload benchmarks that model data sharing patterns of grid RPC applications defies the performance comparison between several data management layers. To address this issue, we develop synthetic grid RPC workload models that abstract data sharing patterns typically needed by grid RPC applications, and investigate the performance characteristics of three data transfer methods in OmniStorage. It shows that selecting an appropriate data transfer mechanism promises to solve an inherent inefficiency in Grid RPC regarding data sharing among a master and workers.

Our contributions are as follows:

- Synthetic workload programs for grid RPC applications are developed to investigate the performance characteristics of several data management layers regarding the data sharing pattern among a master and workers.
- We investigate the performance characteristics of different kinds of data transfer methods in a data management layer using the synthetic workload programs so that an optimal data transfer method for each data sharing pattern is examined.
- Through performance evaluation of OmniStorage, we demonstrate the merits of a program model to decouple the data transmission from the RPC.
- We propose an interface which can utilize a hint information for data sharing patterns in order to allow a program to exploit an optimal data transfer method according to the data sharing pattern needed by the application.

```
/* master program */
int main(){
 double initdata[LEN];
 ...
 for(i = 0; i < n; i++)
  req[i] = OmniRpcCallAsync("foo", LEN, initdata, i);
 ...
 OmniRpcWaitAll(n, req);
}
/* worker program */
Define foo(IN int s, IN double data[s], IN int iter){
 /* main calculation */  }
```

Fig. 3. An example of OmniRPC program

```
/* master program */
int main(){
 double initdata[LEN];
 req = OmstPutData("mydat",initdata,
                    sizeof(double)*LEN,OMST_BROADCAST);
 OmstWait(req);
 for(i = 0; i < n; i++)
  req[i] = OmniRpcCallAsync("foo", i);
 OmniRpcWaitAll(n, req);
}
/* worker program */
Define foo(int IN iter){
 double initdata[LEN];
 req = OmstGetData("mydat",initdata, sizeof(double)*LEN);
 OmstWait(req);
 /* main calculation */   }
```

Fig. 4. An example of OmniRPC program with OmniStorage

The remaining of this paper is organized as follows. Section 2 describes data sharing pattern models typically needed by grid RPC applications and benchmark programs. Section 3 describes a design and implementation of the OmniStorage. The performance and the characteristics of OmniStorage are presented in Section 4. A selection policy of data transfer methods depending on data sharing patterns among a master and workers is discussed in Section 5. Section 6 describes previous work related to the present study. Finally, Section 7 concludes the paper.

2 Data Sharing Pattern for Data Management Layer

2.1 Data Sharing Pattern in Grid RPC Applications

Basically RPC mechanism sends arguments and receives results between a master and a worker. However some grid RPC applications need other data sharing patterns such as broadcasting common initial data, and data transmission between workers. Performance issues for each data sharing pattern can be summarized as follows:

Broadcasting initial data: In case of parametric search type parallel applications, workers will receive common initial data and their own parameters from a master to execute their part of computations at remote nodes. In this case, the master has to send both common initial data and different parameters by every RPC. To address this kind of issue, the OmniRPC provides a data persistence mechanism called "automatic-initializing remote module" to hold data specified by an initialization function of a remote executable module [8]. This avoids multiple transmissions of the same initial data. However, the data must be sent directly from a master to each worker when a remote module is invoked. If the initial data is large, or the number of workers increase, the data transfer from a master would be a bottleneck.

Data transfers between workers due to data dependency in RPC's parameters: If an application has data dependency in parameters of RPCs, in other words, several RPCs have data dependency between input and output parameters, which means output of previous RPC becomes the input of the next RPC. A worker communicates to another worker through a master in order to realize the data transfer between the workers. Therefore the communication to the master will disturb application's scalability especially in case of a grid environment due to a long latency and a poor network bandwidth.

Or if an application performs request sequencing of RPC, the efficiency of data sharing will be improved by optimizing the data sharing between workers.

2.2 Data Sharing Pattern

Data sharing patterns required by grid RPC applications are summarized as follows. Figure 1 shows the overview of these patterns.

W-To-W: W-To-W model is seen at a program which an output of a previous RPC becomes the input of the next PC. W-To-W model is used in the concept of RPC request sequencing so that no unnecessary data is transmitted and all necessary data is transferred. In W-To-W, a worker registers its own data to OmniStorage after that another worker retrieves the data from OmniStorage.

BROADCAST: BROADCAST model is observed at a program to broadcast common initial data from a master to workers. The model is used at many parametric search programs. In BROADCAST model, a master sends one common initial file to all workers.

ALL-EXCHANGE: ALL-EXCHANGE models a program that every worker exchanges their own data files each other for subsequent processing. All workers exchange their own data files each other. In other words, each worker registers one file to OmniStorage after that the worker retrieves files which are registered by the other workers. This model is the worst data sharing pattern in case of a black box program which a worker communicates with other workers randomly.

3 OmniStorage: A Data Management Layer for Grid RPC Applications

3.1 Design of OmniStorage

To handle data sharing patterns described in the previous section, we design a data management layer called OmniStorage on wide area networks to realize the efficient data transfers among workers by decoupling the data transmission from conventional RPC model. Decoupling the data transmission form RPC, OmniStorage framework works as a data repository system for grid RPC applications. Figure 2 shows the overview of OmniStorage. A process can register data with a unique identifier and a hint, which indicates a data sharing pattern, to data repository of OmniStorage. In the data repository of OmniStorage, data is managed by a combination of an identifier and a data entity like "(id, value)". A process can retrieve data by specifying an identifier. The data location is transparent to users. In addition, OmniStorage utilizes a hint information of the data in order to choose a suitable data transfer method according to the data sharing pattern.

OmniStorage APIs consist of a data registering function, `OmstPutData(id, data,size,hint)`, data retrieving functions, `OmstGetData(id,data,size)`, request synchronization functions to access a shared data space. Here, Data handling information are given by a logical sum of hints which indicate data sharing patterns, and data transfer methods with `hint` in OmniStorage data registration API. These hints are summarized as follows:

- Attributes of patterns of data transfer
  - `OMST_POINT2POINT`: Data may be transferred between worker processes.
  - `OMST_BROADCAST`: Data is supposed to be broadcasted to many processes.
  - `OMST_ALL_EXCHANGE`: Data is supposed to be exchanged each other by each worker process.
- Attribute to specify a specific data transfer layer.

In the OmniRPC applications with OmniStorage, shared data among workers are managed by OmniStorage, while none-shared data including a parameter in parametric search applications is managed by OmniRPC's parameters.

Figure 3 and Figure 4 show a typical parametric search application with OmniRPC, and the same application with OmniRPC and OmniStorage, respectively. Both of examples, an initial data are broadcasted to all workers. The master program calls `OmstPutData()` to register data before a worker program accesses the data. Then the worker program calls `OmstGetData()` to access the data. To identify a data, the same identifier "`mydat`" in the name space of OmniStorage is used in both a master and workers. Moreover, a hint of data sharing pattern is specified on the data registration API of `OmstPutData()` so that OmniStorage can select an appropriate data transfer method regarding a data sharing pattern. Here, a hint of broadcast pattern of `OMST_BROADCAST` is specified.

3.2 Implementations

The data cache in OmniStorage is basically handled as the data file. OmniStorage exploits a data transfer method to transfer a cache data file between processes or to access a cache data file. The cache files will be stored on a remote host so that a program can exploit the data locality.

OmniStorage accommodates several data transfer methods to exploit an suitable data transfer method according to a required data sharing pattern. Although in this paper, OmniStorage employs three middleware as data transfer methods which have different kinds of characteristics for performance evaluations as follows:

Omst/Tree: Omst/Tree exploits our file broadcasting middleware taking network topology into account. Figure 5 shows the overview of Omst/Tree. Omst/Tree uses several relay servers between master and workers so that Omst/Tree can reduce the amount of data communications between a master and workers.

Omst/BT: Omst/BT makes use of BitTorrent to be used as a data transfer method for cache files. BitTorrent [9] is a P2P file sharing protocol in order to distribute a large amount of files to many peers efficiently. Figure 6 shows the overview of Omst/BT. When the number of seeder increases according to the progress of file distribution, the load average of each peer gets lower and BitTorrent can achieve efficient data transfers.

Omst/GF: Omst/GF utilize a grid-enabled distributed file system called Gfarm [10]. Gfarm provides a grid file system that can scale up to petascale storage, and realize scalable I/O bandwidth and scalable parallel processing. Cache files on Omst/GF are managed by Gfarm. These caches are accessed by using the Gfarm remote I/O functions. Omst/GF duplicates the cache files on several nodes in order to improve the scalability of data transfer. For example, if broadcast type data transfer is used in a program, Omst/GF duplicates two more cache files on file system nodes.

4 Performance Evaluation

The basic performance and the characteristics of OmniStorage implementations are investigated using three synthetic benchmark programs by data sharing patterns needed by grid RPC applications. The benchmark programs consist of **W-To-W**, **BROADCAST**, and **ALL-EXCHANGE** based on data sharing patterns as discussed at Section 2. Note, we do not mention the basic performance of the middleware used in OmniStorage, such as BitTorrent and Gfarm. As for these basic performances, refer the reports [11,12,13].

Four clusters connected by different networks are used for performance evaluation. Figure 7 shows cluster configurations and the measured network performance. Each master program of benchmark programs is executed on cTsukuba, and a worker program is assigned per computational node in the clusters. Here, each measured data in the performance evaluations is a mean value of five trials. The software components of OmniRPC 1.2, libtorrent 0.9.1, Azureus 2.5.0.2, Gfarm 1.4, Boost C++ Library 1.33.1, and Java2 SE 1.4.2 are used.

To conduct this performance evaluation, two experimental settings with 16 nodes, where the network configurations between two clusters are different, are configured as follows:

CASE1: Two clusters are in the same network — Using both eight nodes on Dennis cluster and eight nodes on Alice cluster

CASE2: Two clusters are connected by WAN — Using both eight nodes on Dennis cluster and eight nodes on Gfm cluster.

Figure 8 shows the elapsed time of W-To-W benchmark in case of machine configuration of CASE1 and CASE2. In spite of the experimental configuration, Omst/GF achieves better performance than other data transfer methods. In case of Omst/GF and Omst/BT, a worker can directly communicate with another worker so that the efficiency of data transfer between two workers may be improved. On the other hand, to perform W-To-W pattern sharing using only OmniRPC system, two more RPCs, one to send data from a worker to a master, and one to send data from the master to another worker, are required. These sharing patterns cause bottlenecks to improve the performance. Moreover if the necessary number of W-To-W type sharings increases, the efficiency of data transfers would go from bad to worse.

Although Omst/BT could not achieve better performance than using only OmniRPC system. When the number of running workers is small, the BitTorrent protocol cannot perform data transfers efficiently. Moreover, Omst/BT needs pre-procedures, such as creating a torrent file and uploading the torrent file, before the data sharing among a master and workers starts so that the performance of Omst/BT is degraded.

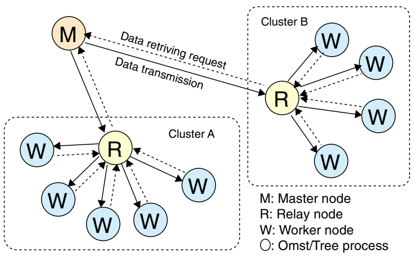

Fig. 5. Omst/Tree implementation

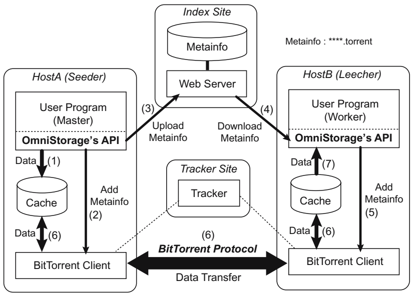

Fig. 6. Omst/BT implementation

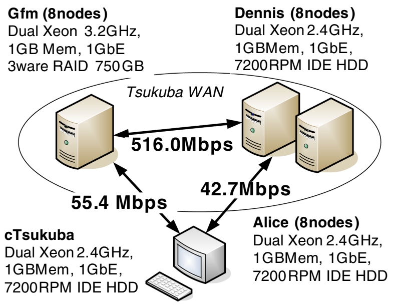
Gfm (8nodes)
Dual Xeon 3.2GHz, 1GB Mem, 1GbE
3ware RAID 750GB
Dennis (8nodes)
Dual Xeon 2.4GHz, 1GBMem, 1GbE, 7200RPM IDE HDD
Tsukuba WAN
516.0Mbps
55.4 Mbps
42.7Mbps
cTsukuba
Dual Xeon 2.4GHz, 1GBMem, 1GbE, 7200RPM IDE HDD
Alice (8nodes)
Dual Xeon 2.4GHz, 1GBMem, 1GbE, 7200RPM IDE HDD

Fig. 7. Experimental platform

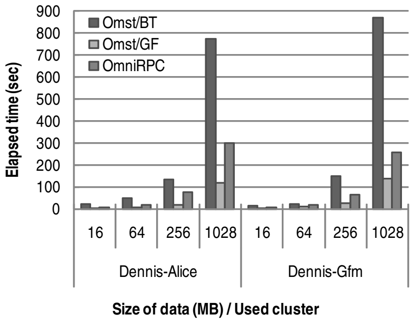

Fig. 8. Elapsed time of W-To-W benchmark

Figure 9 shows the elapsed time of BROADCAST on two types of machine configuration. Regardless of the data size, Omst/Tree achieves better performance in most cases. In case of two clusters connected by high speed network, Omst/BT can achieve approximately the same performance on the Omst/Tree. Particularly when the data size is 1024MB, Omst/BT obtains better performance than Omst/Tree does. In Omst/BT, a data file is divided many small pieces, and each piece is transferred in parallel to other processes so that the efficiency of data transfer is improved. Omst/GF performance is degraded because of the increase of number of data communication between a master and a worker by an inadequate scheduling about a host selection in Gfarm.

Figure 10 shows the elapsed time of ALL-EXCHANGE benchmark program in case of CASE1 and CASE2. Omst/GF succeeds twice faster than Omst/BT in all data size and two cluster configurations. In case of the original OmniRPC system, a master sends a large amount of data many times so that the data sharing between the master and a worker becomes a serious bottleneck, as a result, the performance is degraded. Some factors of performance bottleneck on Omst/BT are caused by both inadequate algorithms, such as the way to selection peer on a tracker and choking algorithm on a peer, and parameter configurations of BitTorrent, such as the limit bandwidth and the limit of connections on a data seeder. However there may be an opportunity to improve the performance by optimizing these parameters or replacing the algorithm.

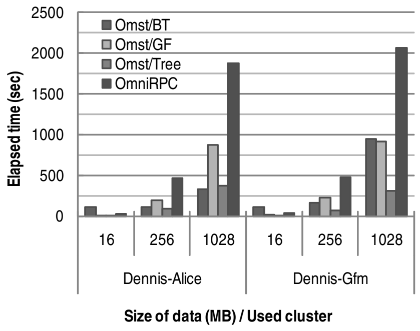

Fig. 9. Elapsed time of BROADCAST benchmark

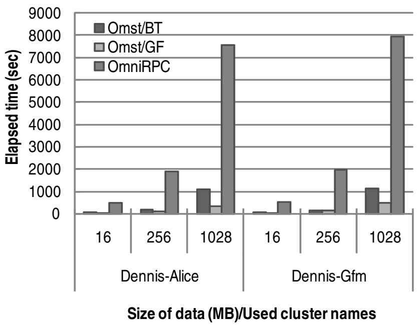

Fig. 10. Elapsed time of ALL-EXCHANGE benchmark

5 Discussion for an Optimal Data Transfer Method

We discuss for an optimal data transfer method of OmniStorage according to three data sharing patterns from the basic performance evaluation.

W-To-W: Omst/GF is preferred in case of a data sharing pattern of W-To-W. One of reasons may come from the environment in the performance evaluation such that there are too small number of workers to get merits of BitTorrent protocol.

BROADCAST: Omst/Tree is preferred for BROADCAST when the network topology of a grid environment is known. In case that the network topology is not known beforehand, Omst/BT is preferred because the BitTorrent protocol aims to be used in a peer-to-peer environment where the network topology is unknown. In addition, when a user exploits more than 1000 nodes, Omst/Tree requires more complex configurations to run than Omst/BT. Whence Omst/BT is suitable in such case.

ALL-EXCHANGE: A suitable data transfer method is Omst/GF in case of ALL-EXCHANGE. In this evaluation, BitTorrent parameters is not optimized in order to improve the Omst/BT's performance. However, there is a chance to improve the performance of Omst/BT.

We move to discuss the merit of exploiting hint information of data sharing pattern in order to achieve the efficient data sharing. The hint information of the data tells the OmniStorage which data sharing pattern is required and how the data is used in a remote process. Therefore, OmniStorage can select an optimal data transfer method which is the most suitable for the data sharing pattern. In addition, OmniStorage can transfer the data efficiently taking the network topology into account. Suppose that OmniStorage does not handle the data hint information, OmniStorage use W-To-W data sharing pattern, and cannot accomplish the efficient data transfers in case that data broadcasting is required. By exploiting some hint information for data, OmniStorage can exploit an optimal data transfer method depending on data sharing patterns so that it can achieve the efficient data transfer in terms of scalability and performance.

6 Related Work

DIET [14] implemented a data management layer called Data Tree Manager (DTM) to avoid multiple redundant transmissions of the same data from a master to workers using a data cache mechanism, and to provide a persistent data access mechanism without any information of data location by using an identifier. This approach mainly focuses on how to handle the persistent data shared among workers. DTM may reduce the amount of data transfer using the data cache mechanism and finding the shortest path to the data from a consumer. On the other hand, OmniStorage focuses on improving both the efficiency of data transfer among workers and the scalability of grid RPC applications. DIET DTM supports W-To-W data sharing pattern, but it does not support other data sharing models and mechanisms to share the data efficiently. OmniStorage can select an optimal data transfer method from several data transfer methods taking required data sharing pattern into account so that OmniStorage improves the efficiency of data transfer among a master and workers including in case of BROADCAST and ALL-EXCHANGE patterns.

NetSolve [2] integrates Distributed Storage Infrastructure (DSI) named Internet Backplane Protocol [15] to control the placement of data that will be accessed by workers so that a master can reduce the times of the same data transfer to workers. However, data in DSI is still explicitly transferred to/from the storage servers at the application level. That means, a master should know both which node stores the data and which node are closer to the worker before the program runs. On the other hand, OmniStorage can adapt to dynamic environment since OmniStorage provides high-level data access APIs without data location information, such as a node name, and optimizes the data transfer automatically. [6] gave a performance result of NetSolve with DSI using a matrix multiply, but the detailed performance evaluation based on data sharing patterns hasn't been presented yet.

Tanimura el al. proposed a task sequencing which allowed direct data transfer of only file type parameter between RPC workers using Gfarm distributed file system as a data repository on Ninf-G [16]. On the other hand, OmniStorage can handle both file and array data and optimize the data transfer by selecting a suitable data transfer method with hint information.

Batch-Aware Distributed File System (BAD-FS) [17] and Stork [18] aim to orchestrate I/O-intensive batch workloads on remote clusters. They manage data placement, data replication according to job requests submitted to a job scheduler. They statically optimize data transfer beforehand the execution. However, OmniStorage focuses on run-time data transfer instructed by an application.

7 Conclusion and Future Work

OmniStorage is designed and implemented to realize a data management layer that augments functionality of the Grid RPC model in order to decouples the data transmission from RPC mechanism aiming to achieve flexible and efficient data transfer and choose a suitable data transfer method of OmniStorage. The basic performance of three implementations is investigated using synthetic benchmark programs based on data sharing patterns needed by grid RPC applications. OmniStorage achieved better performance than the original OmniRPC system in terms of both scalability and efficiency of data transfer. We have demonstrated the merits of a programming model which decouples the data transmission from the RPC. Moreover, taking a hint information of data sharing patterns into account, OmniStorage accomplishes both high performance and high scalability of applications. As our future work, we will optimize some parameters of OmniStorage especially parameters for Omst/BT.

References

1. Nakada, H., Sato, M., Sekiguchi, S.: Design and Implementations of Ninf: towards a Global Computing Infrastructure. Future Generation Computing Systems 15(5-6), 649–658 (1999)
2. Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z., Vadhiyar, S.: Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee (June 2002)
3. Caron, E., Desprez, F.: Diet: A scalable toolbox to build network enabled servers on the grid. International Journal of High Performance Computing Applications 20(3), 335–352 (2006)
4. Sato, M., Boku, T., Takahashi, D.: OmniRPC: a Grid RPC system for Parallel Programming in Cluster and Grid Environment. In: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, pp. 206–213 (2003)
5. Aida, Y., Nakajima, Y., Sato, M., Sakurai, T., Takahashi, D., Boku, T.: Performance Improvement by Data Management Layer in a Grid RPC System. In: Proceedings of the First International Conference on Grid and Pervasive Computing, pp. 324–335 (2006)
6. Beck, M., Arnold, D., Bassi, A., Berman, F., Casanova, H., Dongarra, J., Moore, T., Obertelli, G., Plank, J., Swany, M., Vadhiyar, S., Wolski, R.: Middleware for the use of storage in communication. Parallel Comput. 28(12), 1773–1787 (2002)
7. Del-Fabbro, B., Laiymani, D., Nicod, J.-M., Philippe, L.: Dtm: a service for managing data persistency and data replication in network-enabled server environments: Research articles. Concurr. Comput.: Pract. Exper. 19(16), 2125–2140 (2007)
8. Nakajima, Y., Sato, M., et al.: Implementation and performance evaluation of CONFLEX-G: grid-enabled molecular conformational space search program with OmniRPC. In: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 154–163 (2004)
9. BitTorrent, `http://www.bittorrent.com/`
10. Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., Sekiguchi, S.: Grid datafarm architecture for petascale data intensive computing. In: Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 102–109 (2002)
11. Qiu, D., Srikant, R.: Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: Proceedings of The 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 367–378 (2004)
12. Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Some observations on bittorrent performance. In: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 398–399 (2005)
13. Ogura, S., Matsuoka, S., Nakada, H.: Evaluation of the inter-cluster data transfer on Grid environment. In: Protocols of 3rd International Symposium on Cluster Computing and the Grid, pp. 374–381 (2003)
14. Del-Fabbro, B., Laiymani, D., Nicod, J.-M., Philippe, L.: Data management in grid applications providers. In: The First International Conference on Distributed Frameworks for Multimedia Applications (DFMA 2005), pp. 315–322 (2005)

15. Bassi, A., Beck, M., Moore, T., Plank, J.S., Swany, M., Wolski, R., Fagg, G.: The Internet Backplane Protocol: a study in resource sharing. Future Generation Computer Systems 19(4), 551–561 (2003)
16. Tanimura, Y., Nakada, H., Tanaka, Y., Sekiguchi, S.: Design and Implementation of Distributed Task Sequencing on GridRPC. In: Proceedings of the 6th IEEE International Conference on Computer and Information Technology (2006)
17. Bent, J., Thain, D., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Livny, M.: Explicit control a batch-aware distributed file system. In: Proceedings of the 1st Symposium on Networked Systems Design and Implementation, p. 27 (2004)
18. Kosar, T., Livny, M.: Stork: Making data placement a first class citizen in the grid. In: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004), pp. 342–349 (2004)