9th IEEE/ACM International Symposium on Cluster Computing and the Grid

A Model-Based Algorithm for Optimizing I/O Intensive Applications in Clouds using VM-Based Migration

Kento Sato†, Hitoshi Sato†, Satoshi Matsuoka†‡
† Tokyo Institute of Technology
2-12-1-W8-33, Ookayama, Meguro-ku, Tokyo 152-8552
‡ National Institute of Informatics
National Center of Sciences 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430
kent@matsulab.is.titech.ac.jp, {hitoshi.sato,matsu}@is.titech.ac.jp

Abstract

Federated storage resources in geographically distributed environments are becoming viable platforms for data-intensive cloud and grid applications. To improve I/O performance in such environments, we propose a novel model-based I/O performance optimization algorithm for data-intensive applications running on a virtual cluster, which determines virtual machine(VM) migration strategies, i.e., when and where a VM should be migrated, while minimizing the expected value of file access time. We solve this problem as a shortest path problem of a weighted direct acyclic graph (DAG), where the weighted vertex represents a location of a VM and expected file access time from the location, and the weighted edge represents a migration of a VM and time. We construct the DAG from our markov model which represents the dependency of files. Our simulation-based studies suggest that our proposed algorithm can achieve higher performance than simple techniques, such as ones that never migrate VMs: 38% or always migrate VMs onto the locations that hold target files: 47%.

1. Introduction

Federated storage resources in geographically distributed environments are becoming viable platforms for data-intensive cloud and grid applications, since they can provide much larger amounts of storage resources with abundant computational power than those of typical single-site environments. In such environments, achieving high I/O performance is significant problem, since it can be affected by inter-site network bandwidth. Previous studies have focused on data management techniques such as file replication and caching to reduce costly network file accesses [15]. However, these techniques introduce a large amount of file transfer and storage consumption because of vast sizes of target files, which affects efficient resource utilization.

Virtual clusters [12, 9], which are effectively the resource allocation methods for clouds such as the Amazon EC2 [1], can solve this difficulty by migrating VMs onto close locations to target files [8]. However, determining optimal VM migration strategies, i.e., which VMs should be migrated to where and when, is a difficult problem, since they depend on the size and locations of target files, the memory size of a VM, and run-time network performance.

To facilitate efficient data access in a virtual cluster, we propose a novel model-based I/O performance optimization algorithm for data-intensive applications, determining VM migration strategies while minimizing file access time, including VM migration time, on the assumption that inter-site network throughputs and the size and locations of target files can be monitored. Our algorithm models the dependency of file accesses as a markov model from which we create a weighted directed acyclic graph (DAG) that represents the transition of VM migration, We then solve this problem as a DAG shortest path problem that minimizes overall expected file access time. Our simulation-based studies suggest that the proposed algorithm can achieve higher performance than simple techniques, such as ones that never migrate VMs: 38% or always migrate VMs onto the sits that hold target files: 47%.

2 Related Work

One of the most effective techniques to improve file I/O performance in geographically distributed environments is to deploy files to closest network proximity of the requesting nodes. In particular, read-access performance can significantly affect data-intensive application turn-around times, since such applications often read large data from data management systems but only write a small amount of results. Therefore, improving I/O performance is especially effective for write-once, read-mostly applications. To improve the I/O performance, there is file replication and caching as most previous studies.

File replication attempts to minimize remote file accesses by creating multiple copies of frequently-accessed popular files distributed across sites. Many data management systems, including replica management systems in grids [13, 15], distributed file systems [2, 7], employ this approach.

File caching in data management systems is effective approach to improve I/O performance. For example, optimally selecting files to cache has to consider how likely they are used again in both caching situations. However, one of the difficulties that is unique to data manage systems is that optimal caching also depends on where cached files are originally located. For instance, caching files that are available close to requesting nodes might not contribute to reducing application run times. In contrast, files that are used infrequently but are located far away from requesting nodes might be worth caching, because avoiding future accesses to such files can reduce access times substantially. Therefore, optimal caching strategies need to consider not only access patterns but also wide-area network properties such as latencies and bandwidths.

For these reasons, determining optimal replication strategies, i.e., which files should be replicated to which sites, is a difficult problem, mainly because they depend on application of file access patterns that may only be observed at application run times. These file replication and cache techniques are effective to improve file I/O performance. However, these techniques have several limitations in data-intensive applications, which introduce a large amount of file transfer and storage consumption because of vast sizes of target files.

Virtual clusters [12, 9], which are a virtualized computing clusters that consist of underlying multiple physical computing cluster nodes on a widely distributed environment inter-connected with physical or overlay networks, can solve this difficulty by migrating VMs onto close locations to target files [8]. Therefore, it is required to consider not only migration of data but also migration of processes, i.e., virtual machine, for further I/O performance improvement in geographically distributed environments, which are the underlying platform for clouds and grids. There have been a number of proposals in VM migration techniques [8, 11], however these techniques focus on VM migration for just management flexibility or power efficiency in wide area networks. There are also research works by using VM migration to increase the performance of virtual cluster [14], which transparentlly migrate MPI processes running on virtual machines for load balancing. However, as far as we have researched, we have not been able to encounter a concrete algorithm for run-time migration of multiple jobs using VMs for optimizing parallel workloads in order to optimize I/O in a wide-area distributed environment.

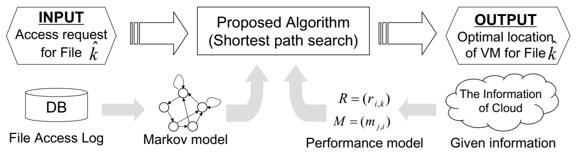


Figure 1. The overview of our proposed algorithm

3 Proposal

Our algorithm determines VM migration strategies, i.e., which VM should be migrated to where, while minimizing overall expected file access time, on the assumption that the inter-site network throughputs and the size and locations of target files are given. Figure 1 illustrates the overview of our proposed algorithm. We create a markov model that represents the dependency of file accesses between files using collected access profiles. Then, we create a weighted DAG, where the weighted vertex represents a location of a VM and expected file access time when the VM accesses a target file, and the weighted edge represents a migration of a VM and time. Then, we solve this problem as a DAG shortest path problem that minimizes overall expected file access time. This section describes the detail of our proposed VM migration strategy.

3.1 Markov Model of File Access Patterns

Our technique requires a sequence of file accesses to estimate the dependency of accesses between files. To represent the access dependency between files, we construct a markov model that represents the probability of access transitions from one file to another from monitored trace, *File Access Log* shown in Figure 1. We use this model to estimate expected file access time to target files. For example, let the markov model be constructed as described in Figure 2. In this model, the successor of *File 1* is expected *File 2* in 90% and *File 3* in 10%. We can describe this state transition diagram as a stochastic matrix, which represents a markov chain over a finite state space. If the probability of transition from *File* i to *File* j in 1-step is $Pr(j|i) = p_{ij}$, we can create the stochastic matrix, $P = (p_{ij})$. The stochastic matrix of the markov chain in Figure 2 can be represented

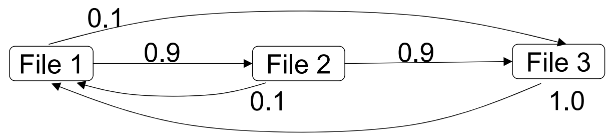


Figure 2. Example of Markov model

as follows:

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \left(\begin{array}{ccc} 0 & 0.9 & 0.1 \\ 0.1 & 0 & 0.9 \\ 1.0 & 0 & 0 \end{array} \right) \end{array}$$

Note that the probability of transition from *File* i to any files must be 1, since the summation of the : $\sum_j p_{ij} = 1$.

Here, we can further extend this stochastic matrix of 1-step transition from *File* i to *File* j to s steps. We denote the probability of transitions from *File* i to *File* j in s-step as $p_{ij}^{(s)}$. For example, the stochastic matrix of 2-step transitions from *File* i to *File* j in Figure 2 can be represented as follows:

$$P^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \left(\begin{array}{ccc} 0.19 & 0 & 0.81 \\ 0.9 & 0.09 & 0.01 \\ 0 & 0.9 & 0.1 \end{array} \right) \end{array}$$

By using this markov model, we estimate future file accesses from a target file to another file.

3.2 Optimal Relocation of VM

Our algorithm determines optimal VM migration strategy, i.e., whether we should migrate VM or not to where and when. VM migration strategy determined by our algorithm selects optimal locations that minimize expected access time of following file sequences. Here, we assume that inter-site network bandwidths are given and the size and locations of files are collected from data management systems. Furthermore, we assume that files have their ID numbers from 1 to K, and file access transitions obey our markov model described in Section 3.1. We determine optimal locations for file accesses by solving shortest path problems of DAG. Figure 3 illustrates a general topology of the DAG constructed in our algorithm, in which vertexes are grouped in several stages from *Stage* 1 to *Stage* S. In our algorithm, VM accesses just one file in each stage. The vertexes of each stage represent locations of a VM, where M is the number of sites, and their weights represnt the expected file access time from the current site to the target file location. On the other hand, the edges represent transitions of a

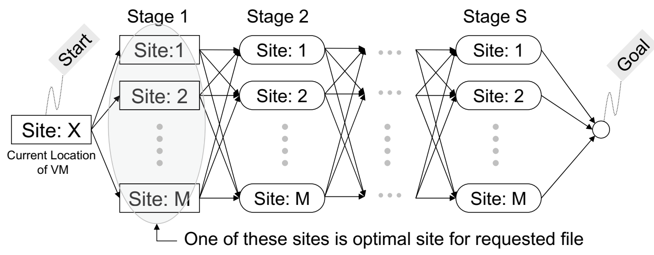


Figure 3. The general topology of a DAG

VM from the current site to the next location in next stage, and their weights represent VM migration times.

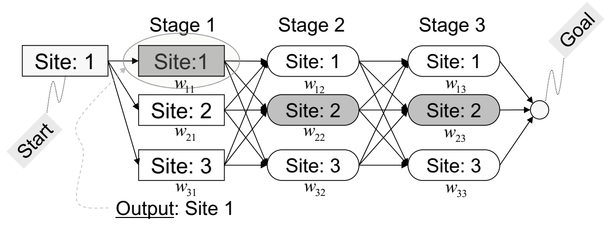
Let the access time of *File* $k \in \{1, \cdots K\}$ from the *Site* $i \in \{1, \cdots, M\}$ be r_{ik} (the detail will be described in Section 3.3), and the requested access file be *File* $\hat{k}$. We denote the expected file access time from *Site* i to the location of *File* $\hat{k}$ at *Stage* $s \in \{1 \cdots S\}$ as follows:

$$w_{is} = \begin{cases} r_{i\hat{k}} & \text{if } s = 1 \\ \sum_{k=1}^{K} p_{\hat{k}k}^{(s-1)} \cdot r_{ik} & \text{Otherwise} \end{cases}$$

where $p_{\hat{k}k}^{(s)}$ denotes the probability of file access transitions from *File* $\hat{k}$ to *File* k in s-steps, which is derived from the stochastic matrix, P^s.

On the other hand, let the VM migration time from *Site* j to *Site* i be m_{ji}. Note that we introduce a *goal* vertex as a end point of the shortest path, to which the edge weights from all vertexes on Stage S represent 0. In addition, we have to determine the number of stages, S, where S denotes the number of files to be considered for solving a shortest path search. In case of considering a large number of stages, it may take a long time to solve a shortest path search for optimal location because of a large number of vertexes, while in case of considering few stages, it may increase the total file access time due to the inaccurate location estimation. We employ the number of stages, S, at which the transitions of file accesses return to the requested file again, i.e., File $\hat{k}$.

Figure 4 illustrates the example of the constructed DAG, in which file access patterns obey the markov model in Figure 2 and the VM located on *Site 1* in 3 sites accesses *File 3* at first. When the VM transitions as follows: (Stage 1,Site 1)$\mapsto$(Stage 2,Site 2)$\mapsto$(Stage 3,Site 2), the expected total file access time represents $m_{11} + w_{11} + m_{12} + w_{22} + m_{23} + w_{33}$, which includes the summation of all vertex weights and edge weights in the path. Therefore, the best VM migration storategy in 27(=3^3) ways is shortest path from *Start* to *Goal* vertex. Note that m_{ii} represents 0, since this means not migrating the VM. In our experiments, we solve a short-



$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} = \begin{pmatrix} r_{13} & 1.0\cdot r_{11}+0\cdot r_{12}+0\cdot r_{13} & 0\cdot r_{11}+0.9\cdot r_{12}+0.1\cdot r_{13} \\ r_{23} & 1.0\cdot r_{21}+0\cdot r_{22}+0\cdot r_{23} & 0\cdot r_{21}+0.9\cdot r_{22}+0.1\cdot r_{23} \\ r_{33} & 1.0\cdot r_{31}+0\cdot r_{32}+0\cdot r_{33} & 0\cdot r_{31}+0.9\cdot r_{32}+0.1\cdot r_{33} \end{pmatrix}$$

Figure 4. Example of the constructed DAG. In this example, our algorithm outputs Site1 as a optimal VM location for accessing File 3

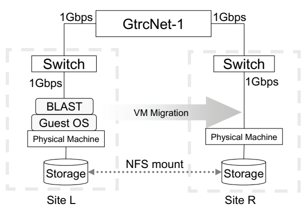


Figure 5. Experiment environments

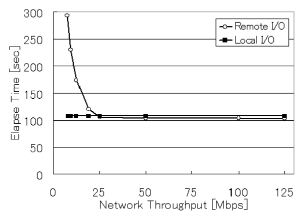


Figure 6. File size: 2GB

est path problem by using Dijkstra's algorithm, but other heuristical methods can be employed for large systems. Our technique conducts the optimization on every file access, and we employ the first location in the determined shortest path for optimal VM migration. if (Stage 1,Site 1)$\mapsto$(Stage 2,Site 2)$\mapsto$(Stage 3,Site 2) is the shortest path, our algorithm outputs Site 1 (Figure4).

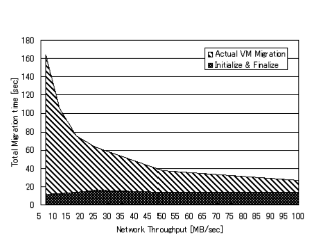


Figure 7. VM Memory Size: 1024MB

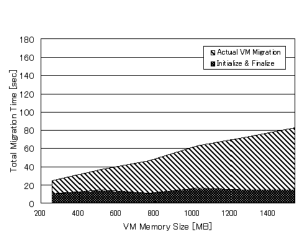


Figure 8. Network Throughput: 25MB/s

3.3 Experimental Setup

We conducted preliminary experiments to create performance models for file access and VM migration times, using two machines connected via network emulator GtrcNet-1 [10] as shown in Figure 5, each of which consists of AMD Opteron 250 (2.4GHz) $\times$ 2, 2GB of memory and gigabit ethernet, running on Linux 2.6.18. We use Xen 3.1.0 [5, 6] for the underlying virtual machine.

First, to create performance model of remote file access time, r_{ik}, we conduct several workloads that sequentially read a file with 2GB size on the remote machine in different network bandwidth settings. Figure 6 compares local and remote read access performance. The x-axis corresponds to the network bandwidth between two machines, while the y-axis corresponds to the elapsed time of read file access. We see that network bandwidth becomes the bottleneck for file access in case of lower network bandwidth. In contrast, local I/O throughput, including memory, HDD and SSD etc, becomes the bottleneck in case the network bandwidth is higher than the local I/O throughput. Thus, we model the file access time r as follows:

$$r = \frac{s}{\min(h_l, h_n)}$$

where s corresponds to the file size to be accessed, h_l to local I/O throughput, and h_n to network bandwidth.

Next, to create performance model of VM migration time, we migrate a VM running an application between two machines under different network throughputs and VM memory sizes. Xen supports *live migration* and *stop-and-copy migration*. We employ *stop-and-copy migration* for accuracy of performance model. We use the BLAST [4] as an execution application, which is a typical bioinformatics tool that finds regions of local similarity between sequences. The total size of dataset to be accessed by the application is 2.03[GB] in this experiment. Figure 7 and Figure 8 show the actual VM migration time, including initialization and finalization operations. We set the memory size of the VM to 1024[MB] in Figure 7, while the network bandwidth to 25[MB/s] in Figure 8. We see that the actual VM migration time exhibits inverse proportion to VM memory size and proportion to network bandwidth. Furthermore, we see that the elapsed time of both initialization and finalization operations for a VM migration are regarded as constant in any VM memory size and network bandwidth cases. Thus, we model the VM migration time m as follows:

$$m = \frac{v}{h_n} + C$$

where v corresponds to the allocated VM memory size, h_n to network bandwidth, and C to the elapsed time for initialization and finalization operations for VM migration.

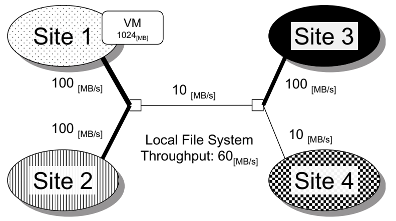


Figure 9. Simulated Grid Environment

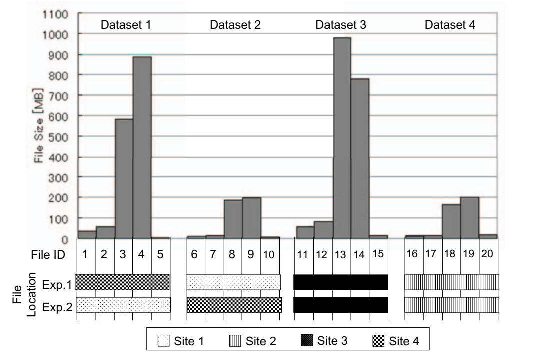


Figure 10. Distribution of file size and location of BLAST dataset

4 Experiment

We conducted performance studies using a simulated geographically distributed environment shown in Figure 9: 4 sites interconnected with 10[MB/s] or 100[MB/s] WANs, and local I/O throughputs of each site to 60[MB/s]. The VM with 1024[MB] memory size is initially located on Site 1. These parameter settings are collected from the given grid environment, the InTrigger [3] resilient grid testbet. Network bandwidths between cluster sites in this environment exhibit 80-100[MB/s] in the best case, 7-10[MB/s] in the worst case. The simulated workload accesses a set of files whose size and location is shown in Figure 10: this behavior is collected from BLAST. We set four datasets and two experimental settings (Exp.1 and Exp.2), which deploy different dataset locations; each dataset is distributed across different sites, and files in a single dataset are located in the same single site. We constructed a markov model of file access transitions from access profiles. We set the elapsed time of initialization and finalization operations for a VM migration to 13[seconds]. We compare our proposed technique (*Proposal*) with two alternative strategies, which are listed below:

- No VM migration is performed (*No migration I/O*). Applications running on a VM always access from the initial location (Site 1 in this experiment).
- VM migration is always performed to the sites that hold target files (*Migration I/O*).

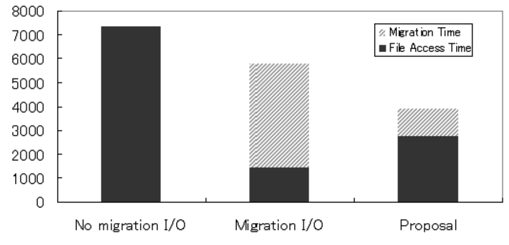


Figure 11. Ex.1: VM Memory size is 1024GB

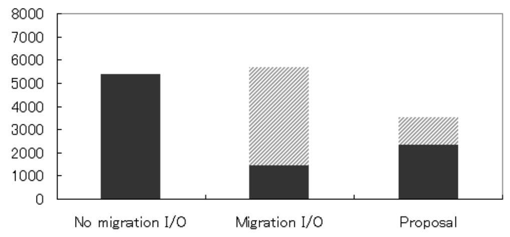


Figure 12. Ex.2: VM Memory size is 1024GB

Figure 11 shows the total read file access time, including local/remote file access and VM migration time with three strategies, when the datasets are distributed as Exp.1 in Figure 10. Compared to the *No migration I/O* strategy, the *Migration I/O* strategy reduces the total file access time by 21% by migrating a VM to sites that holds target files every time. In contrast, our proposed algorithm reduces the total file access time by 47%. Although the actual file access times of our proposed algorithm exhibit longer execution time than that of the *Migration I/O* strategy, our proposed algorithm reduces the total file access time more than that of the *Migration I/O* storategy. This is due to the fact that a large number of VM migration operations occur in the *Migration I/O* strategy, which is a costly operation in WANs. On the other hand, our strategy can avoid unnecessary VM migration and achieve optimal placement for read file accesses.

Figure 12 shows that the total read file access time when the datasets are distributed as Exp.2 in Figure 10. We observe that the total file access time in the *Migration I/O* strategy exhibits longer than that of the *No migration I/O* strategy. This results from the fact that the locations of datasets significantly affect I/O performance of file accesses. In this case, the datasets with large size files are located on the same or close site to the initial location of the

VM (Site 1 in this experiment), which is better condition for the *No migration I/O* strategy than the *Migration I/O* strategy. In contrast, our proposed technique reduces the total file access time by 34% compared to the *No migration I/O* strategy, by 38% compared to the *Migration I/O* strategy, since our strategy avoid costly operations such as remote access to large size files and VM migrations.

In our experiments, we set the number of stages, S, to 5. In case of considering a large number of stages, it may take a long time to solve a shortest path search for optimal location, while few stages, it may cause inaccurate location estimation. However, we observed some irregular situations; in one benchmark, our algorithm using 120 stages exhibits 7.61% total file access time improvement, in contrast, 7.58% total file access time decrease using 100 stages, compared to the one using 5 stages. This is due to that the estimation accuracy is affected by the file accesses of running applications.

Our proposal introduces extra overheads to the original file accesses, which are mainly caused by solving shortest path search problems on every file access request. However, we observed that the elapsed time for solving the optimization problem is more than 270[μ seconds], which would be small or negligible compared to the total file access time of data-intensive application running on geographically distributed environments. Furthermore, we can solve these problems in advance, since the objective shortest path can be determined from the target files; our algorithm considers all possibilities of VM locations. We just look up a corresponding optimal location for the VM from the determined results by using the current VM location and file ID, when a file request is arrived. For example, in the case of $10,000$ files and 14 sites, which is collected from the InTrigger environment, it takes within 37.8[seconds] to determine the optimal VM locations. Further studies may be required if more heuristical approaches would be appropriate for a large number of files, in the millions. We plan to conduct comprehensive analysis to judge the tradeoffs between different shortest path algorithms for our purpose.

5 Conclusion

We have presented a VM relocation algorithm for data-intensive applications on a virtual machine in a geographically distributed environment. Our proposed algorithm determines optimal location of VM to access target file, while minimizing the total expected file access time to files by solving DAG shortest path search problems, on the assumption that the network throughput between sites and the size and locations of target files are given. Our simulation-based studies suggest that the proposed algorithm can achieve higher performance than simple techniques, such as ones that never migrate VMs by 38% or always migrate VMs onto the sites that hold target files by 47%. As a future work, we will consider CPU and memory usages and other VM placements in our proposed algorithm.

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] Hadoop. http://hadoop.apache.org/.

[3] InTrigger. http://www.intrigger.jp/.

[4] NCBI BLAST. `http://www.ncbi.nlm.nih.gov/BLAST`.

[5] Xen Community. http://xen.xensource.com/.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. War eld. Xen and the Art of Virtualization. In *Symposium on Operation System Principles*, 2003.

[7] B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 Protocol, April 2003.

[8] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi. A Storage Access Mechanism for Wide-Area Live Migration of Virtual Machines. In *Summer United Workshops on Parallel, Distributed and Cooperative Processing*, pages 19–24, 2008.

[9] I.Foster, T.Freeman, K.Keahey, D.Scheftner, B.Sotomayor, and X.Zhang. Virtual Clusters for Grid Communities. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 513– 520, 2006.

[10] Y. Kodama, T. Kudoh, R. Takano, H. Sato, O. Tatebe, and S. Sekiguchi. GNET-1:Gigabit Ethernet Network Testbed. `http://projects.gtrc.aist.go.jp/gnet/`.

[11] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and H. Chen. Live and incremental whole-system migration of virtual machine using block-bitmap. In *IEEE International Conference on Cluster Computing*, 2008.

[12] H. Nishimura, N. Maruyama, and S. Matsuoka. Virtual Clusters on the Fly - Fast Scalable and Flexible Installation. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 549–556, 2007.

[13] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *High Performance Distributed Computing*, 2002.

[14] M. Tatezono, H. Nakada, and S. Matsuoka. Mpi environment with load balancing using virtual machine. In *Symposium on Advanced Computing Systems and Infrastructures SACSIS*, pages 525–532, 2006.

[15] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of Data Grids for distributed data sharing, management, and processing. In *ACM Computing Surveys*, volume 38, 2006.