IPSJ Transactions on Advanced Computing Systems Vol. 4 No. 4 110–121 (Oct. 2011)

Regular Paper

Performance Evaluation of a Distributed File System with Locality-Aware Metadata Lookups

Nan Dun,[†1] Kenjiro Taura[†1]
and Akinori Yonezawa[†1]

GMount is a high-performance distributed file system with locality-aware metadata lookups and small installation effort. GMount organizes computer nodes in a decentralized hierarchical overlay to unify separate local file systems into a global shared namespace and achieve locality-aware metadata lookups. GMount offers not only better performance when application executes with considerable data access locality, but also the ability to effortlessly and rapidly enable data sharing among clusters, clouds, and supercomputers. This paper presents performance evaluation of the latest GMount implementation by using both micro-benchmark and real-world data-intensive applications. Experimental results demonstrate that GMount has highly scalable metadata and I/O operation performance when data access locality is common, and the performance of GMount is practically useful for routine data-intensive computing practice.

1. Introduction

Distributed file systems have been used as an indispensable data sharing approach for data-intensive distributed computing. Nevertheless, especially in wide-area computing environments, traditional distributed file systems can encounter four problems due to limited metadata operation scalability and deployment complexity.

High-Latency Metadata Operation Existing distributed file systems usually adopt an architecture consisting of centralized metadata server and multiple data storage server, which can lead to nontrivial problem in high-latency wide-area environments. Specifically, metadata lookup operations will suffer from the high network latency when requesting clients are located far away from the central metadata server. It is particularly inefficient when the target data is stored close to the clients, but the metadata of the target data needs to be retrieved from the distant metadata server. But on the other hand, data-intensive applications have a tendency of *data access locality*, which can be achieved by either application its own inherent structure or file affinity job scheduling from workflow management system [1]. Therefore, it is important that distributed file system should take advantage of data access locality to optimize overall application execution performance by reducing the latency of metadata operations in wide-area environments.

Limitation of Dedicated File Servers Conventional distributed file system usually uses dedicated filesystem servers (both for metadata server and data storage server) to serve file requests from multiple computer nodes. First, this design limits the scalability when the number of computer nodes increases and exceeds the capacity of dedicated file servers. Second, this design does not offer the co-location of data and computation, in which data can be stored and processed on computer nodes locally (e.g., on the local filesystems of computer nodes) to achieve better I/O throughput.

High Setup Cost The installation of most distributed file systems requires sophisticated system knowledge and, sometimes, root privilege. There is an extra setup cost if the filesystem depends on a heavy stack of software. In large-scale environments having many servers, the deployment of distributed file system, usually undertaken by system administrators, is considerable complex, laborious, error-prone, and tedious for general non-privileged users.

Cross-Domain Restrictions Realistic domain policies and restrictions impose additional challenges to installing existing distributed file systems across different administration domains. For example, different domains have different user sets and security policies. Internal dedicated file servers are not allowed to be exported to computer nodes in other domains. Other restrictions of network configurations such as NAT or firewall, can further limit the feasibility to build a common middleware over different administration domains.

Being aware of these problems, we propose GMount — an instantaneously deployable user-level filesystem with locality-aware metadata lookups. By GMount, non-privileged users are able to effortlessly and quickly install a distributed file

†1 Graduate School of Information Science and Technology, The University of Tokyo

system on arbitrary resources they can access. More important, the metadata operation performance of GMount can scale in the wide-area environments by making use of the data access locality of applications. We have illustrated the design and a prototype implementation of GMount in our previous paper [2]. This paper presents the performance evaluation of the latest GMount implementation with several performance enhancements. As a comparison with an existing distributed file system, we particularly study the impact of data access locality on overall scalability and performance of running real-world applications.

2. System Organization

2.1 Overview

Figure 1 shows the architecture of GMount distributed file system. GMount is constructed from a standalone FUSE [3]-based remote file system called SSHFS-MUX [4] (i.e., `sshfsm` in Fig. 1). SSHFS-MUX enables non-privileged users to transparently manipulate files on *multiple* remote machines via *one* mountpoint on local file system (with the union file system semantics [5]), only requiring that users can login to remote machines by SSH.

By a scalable spanning tree based algorithm and two SSHFS-MUX mount operations (i.e., *Union-Mount* and *Cascade-Mount*, see our previous paper [2] for more details), separate computer nodes are organized in a hierarchical overlay in order that: 1) individual namespaces of local filesystems are unified into one global shared namespace, and 2) clients carry out metadata lookups in a locality-aware manner.

In summary, the overlay is constructed as follows. First, separate nodes are grouped according to their network proximity, such as network latency or physical network layout, in order that nodes within the same group are able to communicate with low latency (e.g., group A with node a, c, d, and e). Next, for each group of nodes, one representative node (root node of the tree of this group) is elected to unify the local namespaces of all nodes (e.g., node a for group A and node b for group B). Then, the individual unified namespaces from every group are exported and unified in the same way among representative nodes (i.e., node a and b).

SSHFS-MUX is implemented in about 6000 lines of C codes and requires only standard SFTP server [6] installed beforehand. GMount uses GXP parallel/distributed shell [7] as the distributed loader of GMount to invoke SSHFS-MUX processes in parallel. These two factors allow users to efficiently install GMount in large-scale distributed environments. **Figure 2** illustrates the execution flow of GMount.

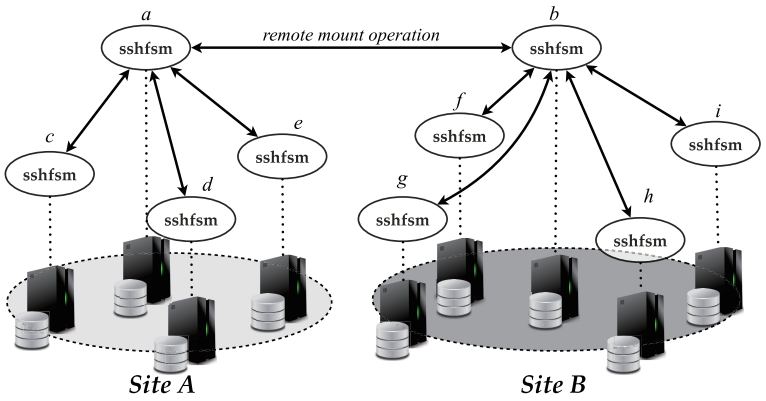


Fig. 1 System architecture of GMount.

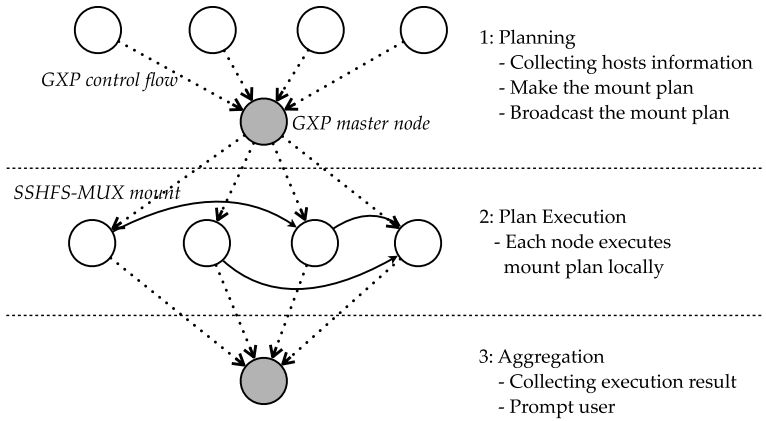


Fig. 2 Execution flow of GMount.

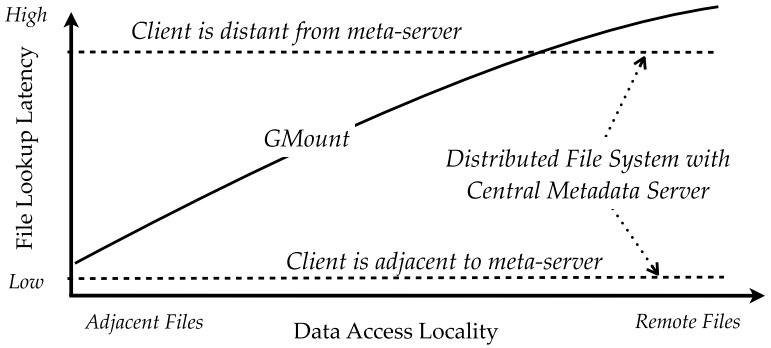


Fig. 3 Comparison of file lookup latency using different architectures.

2.2 Locality-Aware Metadata Lookups

Unlike distributed file systems with central metadata server, there is no centralized metadata repository in GMount and all metadata manipulations are directly performed on the local file systems of the nodes that store the data. Clients lookup files in a location-affinity order over the entire overlay. Specifically, a file lookup first takes place in the client's own local namespace, then this searching is expanded to the namespaces of adjacent nodes (i.e., in the same cluster), and finally floods to other distant nodes until the target data is found. Using Fig. 1 as an example, a client in site A, say node c, first searches the target file within nodes (i.e., a, c, d, and e) of site A, and stops searching if the file is found. Otherwise, the lookup request is forwarded by node a to nodes of group B until the request is satisfied (i.e., file is finally found or does not exist in any node). If there are additional sites, such as another site C and D, node a will forward the search request sequentially, also in a locality-aware manner based on the network affinity between these sites.

Figure 3 summarizes the circumstances when GMount performs better or worse than the distributed file system using central metadata server. For central metadata server approach, the latency of file lookup is determined by the distance between metadata server and the client. For GMount, the latency of file lookup depends on how target file is close to the requesting client. As a result, GMount can outperform central metadata server approach if clients tend to have more data access in adjacent nodes. However, GMount can also fall behind the central metadata server approach when there are more requests for distant files.

The approach of GMount is particularly effective for the kind of distributed applications in which more data is manipulated locally and separately. For example, many data-intensive applications consist of a large amount of jobs to process data separately in parallel, and many intermediate files are created, modified, and finally deleted during the execution. Therefore, synchronizing the updates of these short-life and merely-shared intermediate files can be an overhead for central metadata management, especially in the high-latency environments. However, for those distributed applications in which files traversal (e.g., "`find`" or "`ls -R`" commands) is common or most data is globally shared, GMount could not outperform the central metadata server approach.

2.3 Performance Improvements

Our evaluation of previous GMount implementation has pointed out two major problems that restrict the I/O performance of GMount [2]: 1) limited SSH bandwidth, and 2) congested I/O in hierarchical overlay.

2.3.1 Limited SSH Transfer Rate

Using OpenSSH [6] as data transfer channel can suffer from a low throughput because of limited buffer size and SSH encryption overhead. While the SSH encryption overhead is ignorable because of current fast CPUs, limited transfer buffer will significantly degrade the performance in long fat network. This problem has been well illustrated in previous SSH study [8] and HPN-SSH [9] is a workaround proposed for this problem.

However, applying HPN-SSH requires to patch existing OpenSSH installation in servers, which leads to additional setup effort. By the hints from the "`directport`" option of original SSHFS [10], we propose a simpler and straightforward workaround. We implemented a proxy-like TCP server waiting for client connections instead of using default SSH ports. When a client initials the SFTP request to the server via the raw socket connection, the TCP server accepts the connection, starts up the `sftp-server` process locally, and creates a socket pair to bridge the incoming connection and the input/output streams of `sftp-server` process [6].

This approach honours the underlying SFTP protocol but allows us to use raw

socket instead of SSH. It does not only bypass the SSH limitations but also provides a chance for user-level program to do TCP-tuning or simply leave it to underlying kernel TCP-tuning routines [11]. Though using raw socket for data transfer may raise data security and integrity concerns in untrusted environments, user can return to using default SSH channel, or apply HPN-SSH for better performance without scarifying data security.

2.3.2 Congested I/O in Overlay

The original motivation of constructing the hierarchical overlay is to unify separate namespaces and perform file lookups. Though I/O data traffic can also pass (more specifically, being routed) through overlay, it is inefficient and leads internal node to be a bottleneck.

Therefore, the optimal approach is to enable client to directly access a file for I/O operations if the client knows the storage location of the target file, such as the IP address of server that stores this file. However, since node is mounted via FUSE, which means that we are only able to pass information through standard file system interfaces. Thus, in the case of cascade mount, the client becomes blind to where the target file originally comes from.

It is virtually impractical to modify the interface of FUSE (requires modifying the kernel module) to meet our needs. One possible solution is inspired by the "`use_ino`" option in FUSE [3], which allows SSHFS-MUX program to set the inode number and pass it to `stat()` file system call. The trick is using the inode number to piggyback the node information (i.e., index of node) of the file storage location and let client use this information to establish direct connection for data I/O. Note that the approach does not broke the properties of inode number as long as its uniqueness is guaranteed. As a result, clients establish extra direct connections (separate from the connections used by the overlay) on demand to the nodes that hold the target data for I/O operations, and the overlay is only used for metadata operations.

3. Evaluation

3.1 Experimental Environments

Our experiments used the InTrigger multi-cluster platform [12], consisting of 16 sites of clusters with approximately 400 nodes in total. The sites are geographically distributed in universities and research institutions of Japan. All InTrigger servers have a uniform software installation and configuration: Linux kernel 2.6.26, OpenSSH 5.3p1, FUSE 2.8.3, SSHFS-MUX 1.3 and GXP 3.06. We used 64 available nodes from 5 sites for the evaluation. **Figure 4** shows the configuration of these nodes, with the RTT of network links between sites annotated. Note that the RTT between nodes within the same site is around 0.15 ms.

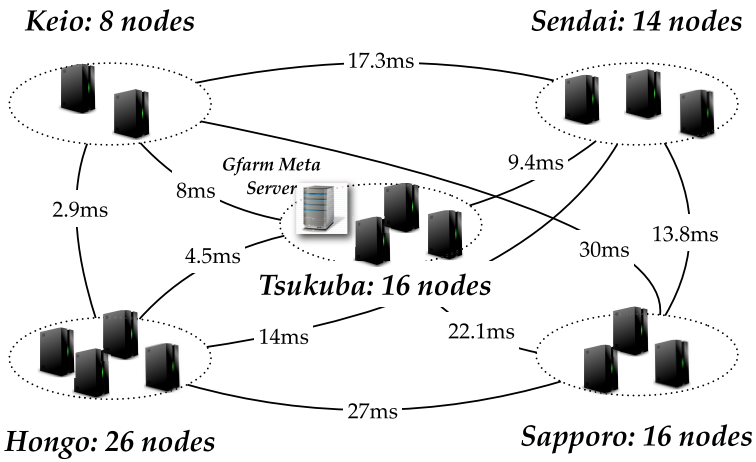


Fig. 4 Configuration of experimental environments.

The micro-benchmark used in experiments is ParaMark [13], which can issue parallel metadata or I/O requests with configurable access pattern from multiple clients to stress the file system.

We compare GMount with Gfarm [1),14)], a wide-area distributed file system with central metadata server. Since Gfarm uses the same strategy as GMount (i.e., co-location of data and computation by federating local file system of computation nodes) and a previous global installation on InTrigger is available for cross-site data sharing, it allows us to conduct a specific metadata operation performance comparison in the same environment. Note that Gfarm metadata server is at *Tsukuba* site.

3.2 Parallel Metadata Operation

We first investigate the impact of data access locality on the overall perfor-

mance. From the client's point of view, metadata access can involve manipulating files that reside on close or distant servers to the client. Here, one node is considered to be *adjacent* to another node if they are in the same site/LAN. Accordingly, we define r_{adj} as the ratio of "the amount of adjacent data access" to "the amount of total data access" of one client, where *adjacent access* is the data access destined for servers that are located in the same cluster/LAN as the requesting client.

Then we synthesize following access pattern to investigate the impact of data access locality on metadata operations performance. First, ParaMark creates and distributes a collection of data files over all nodes. Next, it generates a random access sequence of data files such that this sequence includes a ratio r_{adj} of adjacent access (relative to the requesting client). Then this sequence is used to access data files from the client. The count of each type of metadata operation is 1000 and we use the aggregated throughput of all clients for comparison.

Since Gfarm has one central metadata server located in one site, the metadata operations performance of Gfarm client depends on the client's location relative to the metadata server. Therefore, we differentiate following two configurations: (i) Gfarm in LAN — client is in the same LAN as the metadata server, and (ii) Gfarm in WAN — client is in a different site.

From **Fig. 5**, it reads that the metadata operations performance of Gfarm scales up to around 10000 ops/sec when the number of clients is 16 for most metadata operations except `rmdir`. The parallel metadata operation performance does not scale up significantly when the number of concurrent clients increases from 4 to 16, because the load reaches the capacity of metadata server.

In WAN, Gfarm achieves about 1600 ops/sec peak performance for 32 concurrent clients in our experimental configuration (See **Fig. 6**). Similar results have been reported in another evaluation of Gfarm in InTrigger environment [1)], where `creat` achieves about 1800 ops/sec for 32 concurrent clients and about 2400 ops/sec for 64 concurrent clients. Though the evaluation results are slightly different, they are in the same order of magnitude and the difference can be due to the selection of servers. The high latency between clients and metadata servers leads to overall low metadata operations performance of Gfarm in wide-area environments.

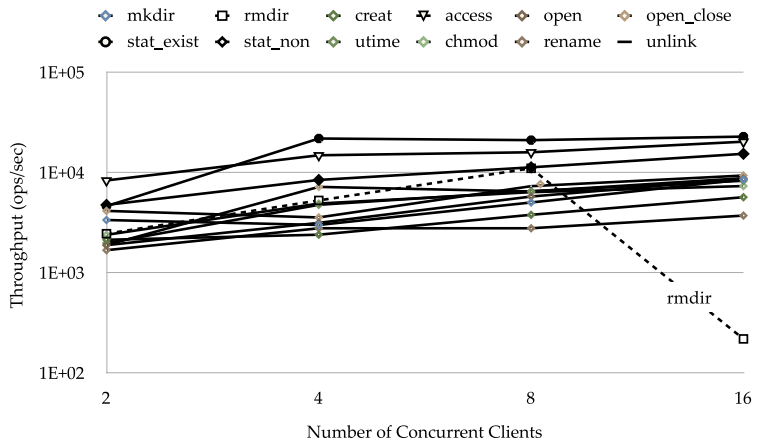


Fig. 5 Parallel metadata operations performance of Gfarm in LAN ($r_{adj} = 1$).

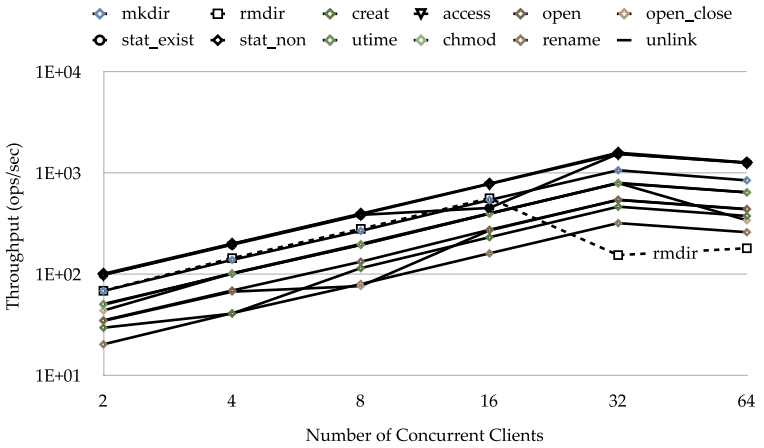


Fig. 6 Parallel metadata operations performance of Gfarm in WAN ($r_{adj} = 1$).

The parallel metadata operation performance of GMount, on the other hand, scales up to an average of 70000 ops/sec for 64 concurrent clients in WAN environment (see **Fig. 7**), which mainly attributes to the locality awareness of metadata operations. Note that it is also comparable to the 100000 ops/sec throughput

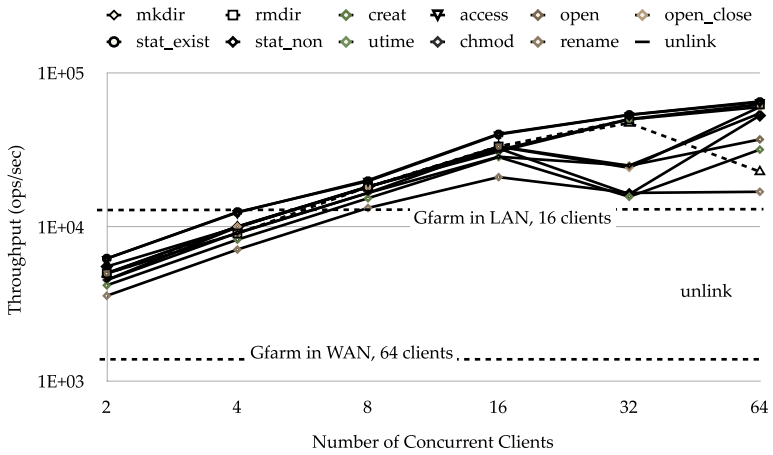


Fig. 7 Parallel metadata operations performance of GMount in WAN ($r_{adj} = 1$).

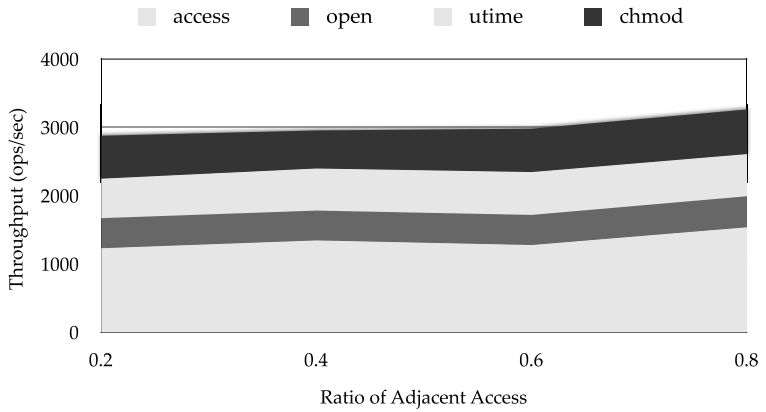


Fig. 8 Parallel metadata operations performance of Gfarm in WAN ($0.2 < r_{adj} < 0.8$).

of Ceph (using multiple metadata servers) for 128 distributed data servers in LAN [15].

Figures 8 and **9** show the evaluation results of Gfarm and GMount, respectively, with 64 concurrent clients when there are data accesses for remote files

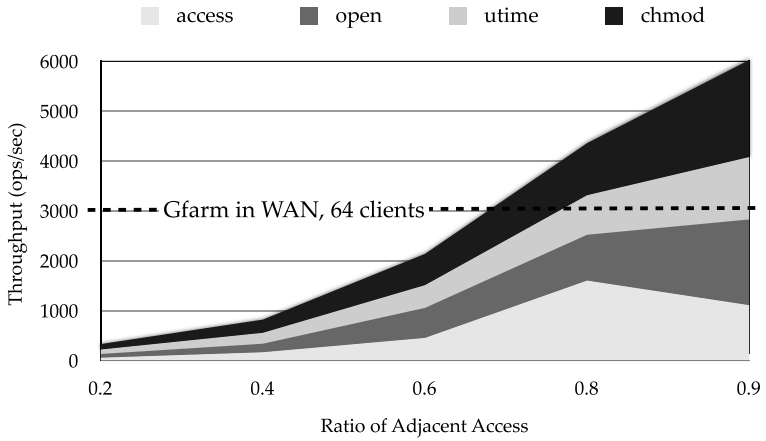


Fig. 9 Parallel metadata operations performance of GMount in WAN ($0.2 < r_{adj} < 0.9$).

(i.e., $r_{adj} \neq 1$). While the performance of Gfarm remains unchanging with the latency between clients and metadata servers, the performance of GMount is sensitive to data access locality of requests. As shown in Fig. 9, GMount achieves an equivalent performance as Gfarm when $r_{adj} \approx 0.65$. However, GMount is about 5x slower than Gfarm when there is only 20% adjacent data accesses, in which GMount has to search local nodes first before looking up in remote nodes, as indicated in Section 2.2.

3.3 Parallel I/O Performance

Before showing overall I/O performance comparison, we present the underlying SSHFS-MUX transfer rate in the WAN environments. Since Gfarm and GMount both establish direct connections for I/O access, point-to-point data transfer rate between nodes over the network (especially over the WAN) is important to overall I/O performance. We choose two distant servers that are connected by high-latency and low-bandwidth WAN links to test data transfer rates of using different file systems. The bandwidth between these two nodes is 150 Mbps and an average of RTT time is 23.6 msec, measured by Iperf [16]. The file size is 256 MB (cache effect does not concern here because network link is a bottleneck) with a variety of block sizes from 4 KB to 16 MB.

Figure 10 presents the comparison results. SSHFS (i.e., equivalent to SSHFS-

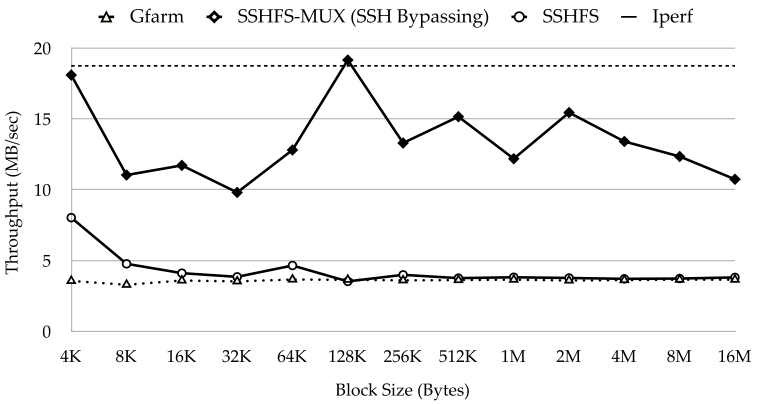


Fig. 10 Comparison of point-to-point data transfer rate in WAN.

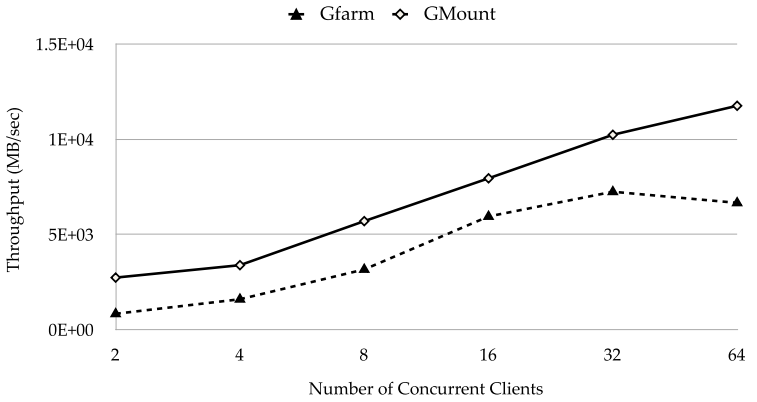


Fig. 11 Comparison of parallel read performance.

MUX without SSH bypassing) and Gfarm are limited to achieve only 25% utilization of the network bandwidth. On the other hand, SSHFS-MUX with SSH bypassing is able to effectively utilize available bandwidth.

Using the same configuration for metadata tests, we evaluate the parallel read and write performance using 2 GB file size and 128 KB block request size for each concurrent client.

Results in **Figs. 11** and **12** show that the performance of Gfarm and GMount

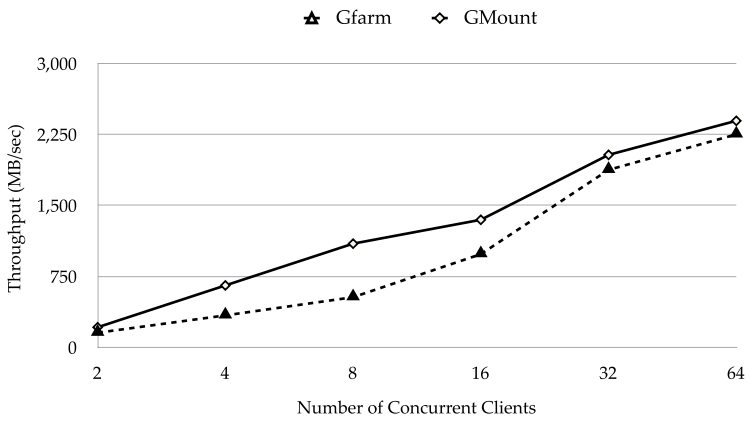


Fig. 12 Comparison of parallel write performance.

both scales with the number of clients, while GMount has an overall 10–20% higher throughput than Gfarm. One possible reason of the I/O overhead of Gfarm is that its data servers need to synchronize metadata updates to metadata server. Comparing with the I/O evaluation of our previous prototype implementation [2] (with default SSH channel and routed I/O traffic), the proposed enhancements (see Section 2.3) significantly boost the I/O throughput and scalability.

3.4 Real-World Applications

Finally, we use two real-world scientific workflow applications to investigate the workflow execution performance by using GMount and Gfarm. We use GXP make [17], a workflow engine based on GNU make, to execute this workflow on the same 64 nodes as in previous experiments. For both GMount and Gfarm, the input data set is initially placed at one node and distributed accordingly by underlying distributed file systems during the execution.

3.4.1 Event Recognition

The event recognition application [18] is to extract and classify biomolecular events mentioned in English texts. Example biomolecular events of interest are an expression of a certain gene, a phosphorylation of a protein, and a regulation of certain reactions. Its execution structure (i.e., workflow DAG) is shown in **Fig. 13**. The input data set consists of 158 input data (i.e., medline XML file),

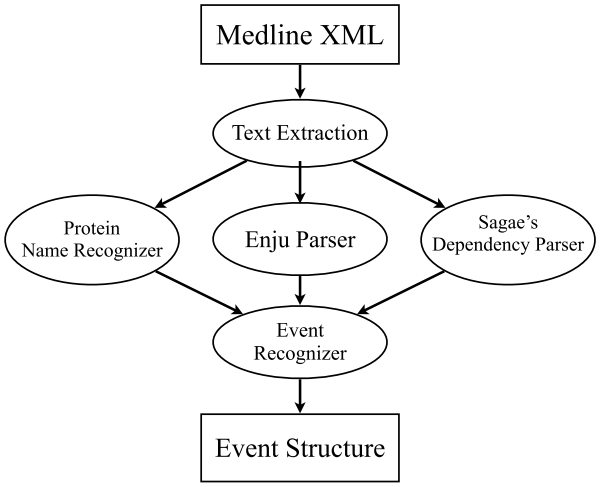


Fig. 13 Workflow DAG of event recognition.

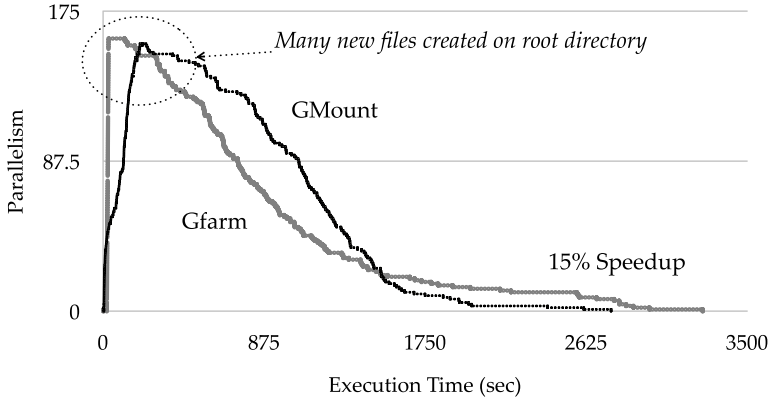


Fig. 14 Execution profiles of event recognition workflow.

where each input data corresponds to a processing unit (the entire DAG chart) shown in Fig. 13. A maximum of 4 concurrent jobs is allowed to run on each node, resulting an overall maximum parallelism of 252.

Figure 14 shows the execution summaries, where the workflow finished in 3257 seconds when using Gfarm and 2759 seconds (or 15% speedup) when using GMount. However, the peak parallelism by using Gfarm (i.e., 159 concurrent jobs) was reached at 28 second, and the parallelism by using GMount (i.e., 156 concurrent jobs) was reached at 198 second. The workflow running on Gfarm has faster initial scheduling performance is because many new files/directories are created under the work directory during the data split and distribution stage at the beginning. In Gfarm, the metadata server can quickly handle these requests because metadata is managed in one place. However, in GMount, a file creation in root directory results a file existence checking in all nodes.

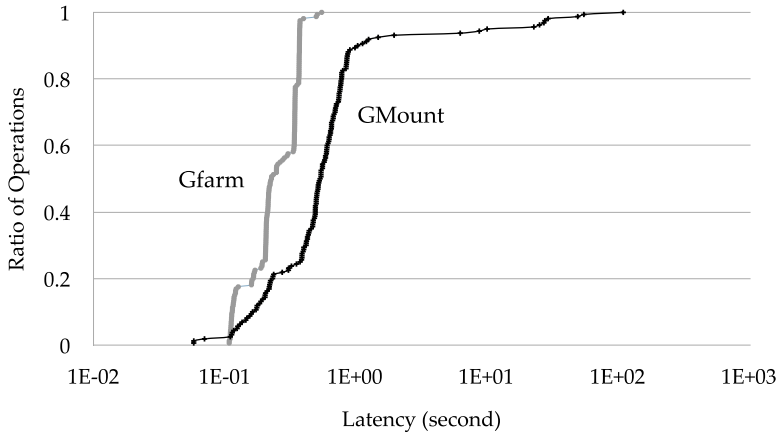


Fig. 15 Latency of file creation in root directory.

To verify this, we extract and collect the execution time of those small jobs that create new files under the working directory. The distributions of execution time of these jobs by using GMount and Gfarm are presented in **Fig. 15**. We found that the file creation in GMount has an average of higher latency than in Gfarm, some of file creation time even are 100 times than the average latency. This is because lookup message floods for concurrent creations on many nodes, resulting saturated data traffic in GMount overlay.

3.4.2 Montage

Montage astronomy scientific application [19] is a popular benchmark that has been widely used in workflow studies [20]. The input data set used in this experi-

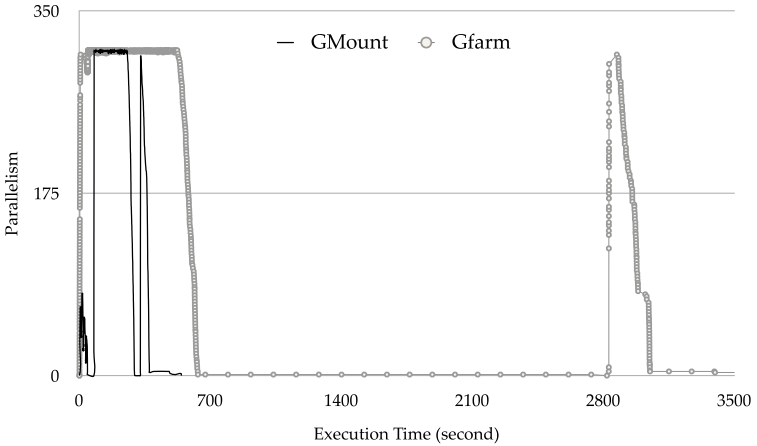


Fig. 16 Execution profiles of Montage workflow.

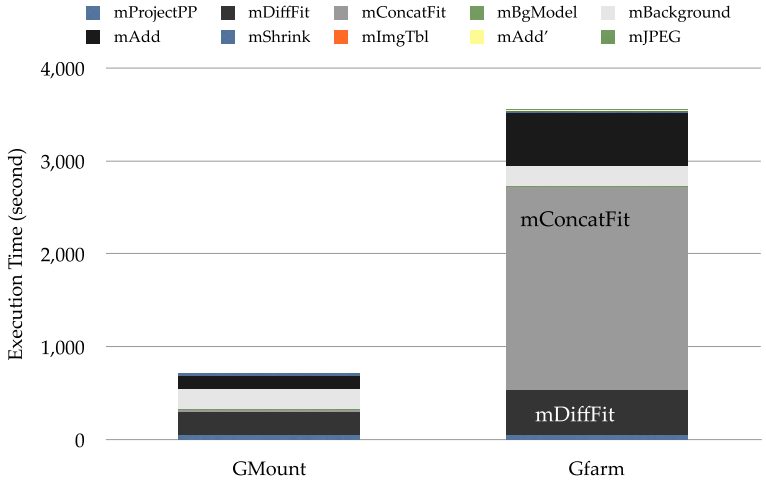


Fig. 17 Comparison of execution time of different phases.

ment includes 609 input files. Note that the input data corresponds to the jobs in the first stages of the entire workflow[⋆1]. Here, we use a smaller scale configuration including 40 nodes from *Keio* and *Hongo* sites since using more nodes does not change the execution structure but only increases the execution time. We still use the existing installation of Gfarm in InTrigger and the metadata server of Gfarm is in *Tsukuba* site. We set a limitation of 8 concurrent jobs on each node, resulting an overall parallelism of 320.

Figure 16 shows execution profiles by using different distributed file systems, where the workflow finished in 547 seconds seconds when using GMount and 3638 seconds when using Gfarm. Note that the execution time can not be compared straightforwardly since computer nodes in *Tsukuba* site are not included, where Gfarm nodes in *Keio* and *Hongo* have to talk to the metadata server in *Tsukuba* site.

Figure 17 demonstrates the comparison of execution time of each phase. Similar as in event recognition workflow, GMount has low performance at the initial phase `mProjectPP` because many files are created under the root directory. The largest difference comes from the phase `mConcatFit`, where 2000 seconds are cost in Gfarm but only 20 seconds are used in GMount. Although the environmental latency for Gfarm is 2–3 times higher than GMount due to the placement of the Gfarm metadata server, it is not enough to make such long execution time. Instead, there is significant high latency in Gfarm when clients establish the first connection to data server for data transfer. In `mConcatFit` phase, the merge process opens and reads *sequentially* many small files (i.e., 913 files, each is 280 Bytes) that are distributed in all nodes, which requires the client to establish connections to every other node one after another and leads to an accumulated long execution time.

4. Related Work

Most existing distributed file systems consist of dedicated file servers and are designed for the highly coupled cluster environments, such as PVFS [21], Lustre [22], GPFS [23], GoogleFS [24], HadoopFS [25], and Ceph [15]. To the best of our knowledge, they have not been deployed for practical usage in multi-cluster environments except GPFS-WAN [26]. Other filesystem evaluation in the wide-area

⋆1 The workflow DAG of Montage is online available at http://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

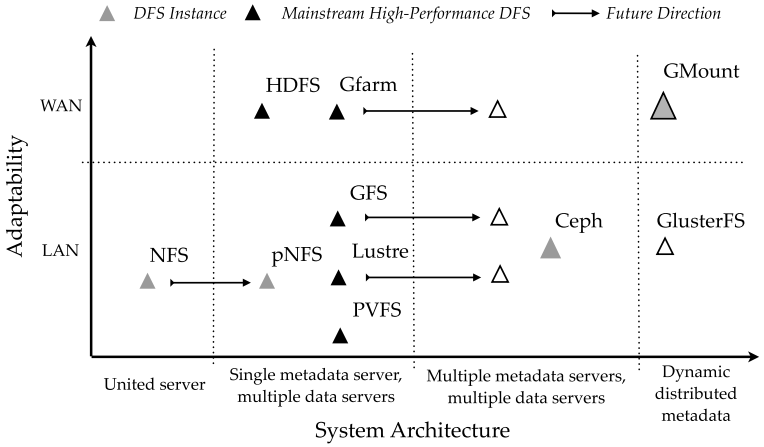


Fig. 18 Architectural comparison of existing distributed file systems.

environments includes Lustre [27),28)], in which the complexity of deployment and low scalability and performance in the wide-area are reported. Ceph [15)] is an experimental peta-scale parallel file system, using *a cluster of metadata servers* to achieve scalable metadata operations. However, the effectiveness of multiple metadata servers has not been studied in multi-clusters environments.

Figure 18 illustrates the architectural comparison of existing distributed file systems. The *decoupling of metadata and file data management*, one of key ideas for distributed file system design, brings out significant scalability improvement of high-performance parallel file systems. PVFS, Lustre, pNFS, GFS, HDFS, and Gfarm all benefit from this principle and thus adopt the "one metadata server, multiple data servers" architecture. However, with the increasing number of clients and data footprint in application, one single metadata server easily reaches its capacity and becomes the bottleneck for further performance scaling. As a result, the *distributed management of metadata* is introduced, and one typical example of this design is Ceph. Besides Ceph, GFS also shows its future plan of using distributed namespace servers [29)]. Lustre and Gfarm also intend to implement multiple metadata servers [14),22)]. Though without many details, GlusterFS [30)] also gives us a hint on using dynamic distributed metadata technique, similar as GMount, to achieve better scalability.

To clarify the differences between GMount and other distributed file systems, we state that: 1) GMount should *not* be considered to be a persistent distributed storage system, it is designed for sharing data rather than storing data, and 2) its usage scenario is different.

First, GMount is an *instantaneous* and user specific file system that is supposed to be constructed on demand and destructed after the usage. It is a complementary distributed file system for data-intensive computing practice, i.e., executing data-intensive applications among cross-domain resources in the wide area, where conventional distributed file systems can not be straightforwardly applied or can not work as well as in the LAN environments.

Second, as the tradeoffs to achieve better performance when there are more data access locality in application execution, GMount uses a weaken cache consistency model and sacrifices the fault tolerance in current implementation. GMount does not have its own cache subsystem or data replication mechanism for I/O or fault tolerance. Data are stored and manipulated directly on the local filesystems where target data stored in remote servers over the network. Since GMount directly harnesses the local file system to store user data, GMount assumes that the data availability is guaranteed by local filesystems of each server.

5. Conclusions and Future Work

GMount is a high-performance distributed file system with locality-aware metadata operations and small installation effort. Evaluation shows that GMount is able to benefit from data access locality of applications to achieve better performance than existing wide-area file system with central metadata server. In addition, with the ability to rapidly and effortlessly unify local file systems of arbitrary resource for global data sharing, GMount is practically useful for general users to conduct their data-intensive distributed computing practice. GMount also shows a novel way of building practically useful distributed file system with simple building blocks and the small implementation.

Our major future work includes the design and implementation of a more sophisticated metadata management to improve the metadata lookup performance by reducing the overhead of global searching when data access locality is not sig-

nificant. We also plan to implement a simpler interface to allow existing workflow management systems to retrieve file storage locations for file affinity scheduling.

GMount is an open-source software and online available at http://sf.net/projects/gxp/ and http://sshfsmux.googlecode.com/.

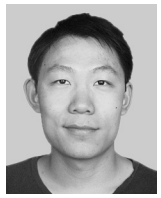
Acknowledgments We would like to thank our colleagues for their valuable suggestions on this work. We also would like to thank the referees for their efforts and comments. This research is supported in part by the MEXT Grant-in-Aid for Scientific Research on Priority Areas project "New IT Infrastructure for the Information-explosion Era."

References

1) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm grid file system, *New Generation Computing*, Vol.28, pp.257–275 (2010).
2) Dun, N., Taura, K. and Yonezawa, A.: GMount: An ad hoc and locality-aware distributed file system by using SSH and FUSE, *Proc. 9th IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid '09*), pp.188–195 (May 2009).
3) Szeredi, M.: FUSE: Filesystem in userspace (online), available from ⟨http://fuse.sourceforge.net/⟩.
4) Dun, N.: SSHFS-MUX (online), available from ⟨http://sshfsmux.googlecode.com/⟩.
5) Wright, C.P., Dave, J., Gupta, P., Krishnan, H., Quigley, D.P., Zadok, E. and Zubair, M.N.: Versatility and Unix semantics in namespace unification, *ACM Trans. Storage*, Vol.2, pp.74–105 (Feb. 2006).
6) OpenSSH (online), available from ⟨http://www.openssh.org/⟩.
7) Taura, K.: GXP: An interactive shell for the grid environment, *Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (*IWIA '04*), pp.59–67, Charlotte, NC, USA (Apr. 2004).
8) Rapier, C. and Bennett, B.: High speed bulk data transfer using the SSH protocol, *Proc. 15th ACM Mardi Gras Conference*, pp.1–7, Baton Rouge, LA, USA (Jan. 2008).
9) HPN-SSH (online), available from ⟨http://www.psc.edu/networking/projects/hpn-ssh/⟩.
10) Szeredi, M.: SSH filesystem (online), available from ⟨http://fuse.sourceforge.net/sshfs.html⟩.
11) Enabling high performance data transfers (online), available from ⟨http://www.psc.edu/networking/projects/tcptune/⟩.
12) InTrigger multi-cluster platform (online), available from ⟨http://www.intrigger.jp/⟩.
13) Dun, N.: ParaMark: Parallel filesystem benchmark (online), available from ⟨http://paramark.googlecode.com/⟩.
14) Gfarm (online), available from ⟨http://datafarm.apgrid.org/⟩.
15) Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E. and Maltzahn, C.: Ceph: A scalable, high-performance distributed file system, *Proc. 7th Symposium on Operating Systems Design and Implementation* (*OSDI '06*), pp.1–7 (2006).
16) Iperf (online), available from ⟨http://sourceforge.net/projects/iperf/⟩.
17) Taura, K., Matsuzaki, T., Miwa, M., Kamoshida, Y., Yokoyama, D., Dun, N., Shibata, T., Choi, S. and Tsujii, J.: Design and implementation of GXP Make – A workflow system based on make, *Proc. IEEE e-Science 2010 Conference* (*e-Science '10*) (Dec. 2010).
18) Miwa, M., Sætre, R., Kim, J.-D. and Tsujii, J.: Event extraction with complex event classification using rich features, *Journal of Bioinformatics and Computational Biology*, Vol.8, No.1, pp.131–146 (Feb. 2010).
19) Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J.C., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, T.A. and Williams, R.: Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking, *International Journal of Computational Science and Engineering*, Vol.4, pp.73–87 (July 2009).
20) Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C. and Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems, *Scientific Programming Journal*, Vol.13, No.3, pp.219–237 (2005).
21) Parallel Virtual File System (online), available from ⟨http://www.pvfs.org/⟩.
22) Lustre file system (online), available from ⟨http://www.lustre.org/⟩.
23) Schmuck, F. and Haskin, R.: GPFS: A shared-disk file system for large computing clusters, *Proc. Conference on File and Storage Technologies*, pp.231–244 (Jan. 2002).
24) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google file system, *Proc. 9th ACM Symposium on Operating Systems Priciples* (*SOSP '03*), pp.29–43, New York, NY, USA (Oct. 2003).
25) Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop distributed file system, *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies* (*MSST '10*) (May 2005).
26) Andrews, P., Kovatch, P. and Jordan, C.: Massive high-performance global file systems for grid computing, *Proc. 2005 ACM/IEEE Conference on Supercomputing* (*SC '05*), IEEE Computer Society, Washington, DC, USA (2005).
27) Cope, J., Oberg, M., Tufo, H.M. and Woitaszek, M.: Shared parallel file systems in heterogeneous linux multicluster environments, *Proc. 6th LCI International Conference on Linux Clusters* (Apr. 2005).

28) Simms, S.C., Pike, G.G. and Balog, D.: Wide area filesystem performance using lustre on the TeraGrid, *Proc. The TeraGrid 2007 Conference* (June 2007).
29) McKusick, K. and Quinlan, S.: GFS: Evolution on fast forward, *Queue*, Vol.53, No.3, pp.42–49 (Mar. 2010).
30) GlusterFS (online), available from ⟨http://www.gluster.org/⟩.

(Received January 28, 2011)
(Accepted May 26, 2011)



Nan Dun is currently postdoctoral researcher at Department of Information and Communication Engineering, The University of Tokyo. He was born in 1980, He received his B.S. from Peking University, M.S., and Ph.D. degrees from The University of Tokyo in 2003, 2007, and 2011. His major research interests include parallel/distributed computing, high-performance computing, and operating systems. He is a member of IEEE.

Kenjiro Taura is associate professor at Department of Information and Communication Engineering, The University of Tokyo. He was born in 1969, and received his B.S., M.S., and DSc degrees from The University of Tokyo in 1992, 1994, and 1997. His major research interests include parallel/distributed computing and programming languages. He is a member of ACM and IEEE.

Akinori Yonezawa was born in 1947. He received his Ph.D. degree in Computer Science from the MIT in 1977. He is professor in Department of Computer Science at The University of Tokyo. He was a member of the Scientific Advisory Board of German National Research Institute of Computer Science (GMD), and served as the president of Japanese Society of Software Science and Technology and a member of the Evaluation and Promotion Committee of Japanese MITI's Real World Computing Program. Since April 2006, he had been the director of the Information Technology Center, The University of Tokyo. Also, he was a member of Microsoft Trust Worthy Computing Academic Advisory Board (TCAAB). He received the AITO Dahl-Nygaard Prize in 2008. He was awarded a "Medal of Honour with a Purple Ribbon of Japan (Shiju-housho)" at Imperial Palace in 2009.