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. Using GMount implies not only better performance when there is 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 latest GMount implementation by using both micro-benchmark and real-world data-intensive applications. Experimental results show that GMount has highly scalable metadata and I/O performance when data access locality is common, and the performance 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. However, particularly in the wide-area computing environments, traditional distributed file systems can encounter several problems in terms of limited metadata operation scalability and deployment complexity.

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

Limitation of Dedicated File Servers Conventional distributed file system usually uses dedicated filesystem servers (both metadata server and data storage server) to server file requests from many computer nodes. First, this design has a problem of limited scalability when the increasing computer nodes exceed the capacity of dedicate file servers. Second, this design does not benefit from the co-location of data and computation, in which data can be stored and processed on computer nodes (e. g., on the local filesystems of computer nodes) to achieve better 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 the 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 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. Dedicated file servers are not exported to computer nodes in other domains. Other restrictions such as network configurations as NAT or firewall, can further limit the

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

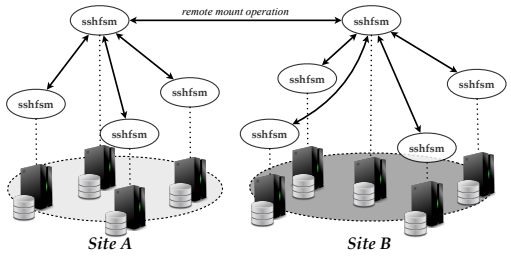

Fig. 1 System Architecture of GMount

feasibility to build a common middleware over different domains.

Targeting these problems, we propose GMount — a 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 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 latest GMount implementation with several performance enhancements. As a comparison with 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 Architecture

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] (denoted as `sshfsm` in Figure 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 details), separate computer nodes are organized in a hierarchical overlay in order that their individual local namespaces/filesystems are unified into one global shared namespace and metadata lookups are carried out in a locality-aware manner.

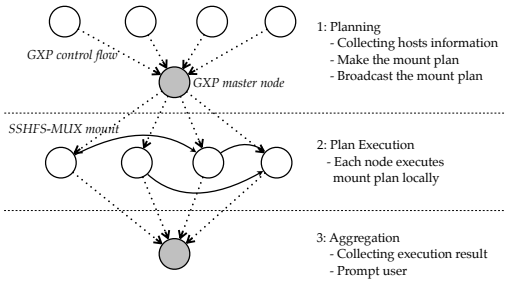

Fig. 2 Execution Flow of GMount

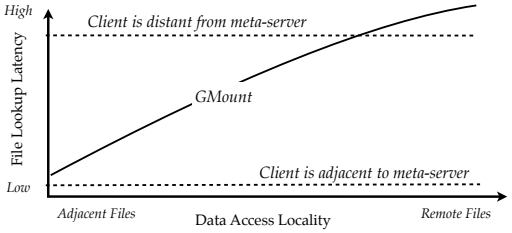

Fig. 3 Comparison of File Lookup Latency using Different Architectures

GMount uses GXP parallel/distributed shell[6] as the distributed loader of GMount to invoke SSHFS-MUX processes in parallel, which allows users to efficiently install GMount in large-scale distributed environments. Figure 2 illustrates the execution flow of GMount.

2.2 Locality-Aware Metadata Lookups

Since there is no a central metadata server, the metadata lookups are performed in a location affinity sequence relative to the requesting client. Specifically, a file lookup first takes place in the client's own local namespace, then in 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 Figure 1 as an example, a client in site A first searches the target file within nodes of site A, and stops searching if the file is found. Otherwise, the client sends addition lookup messages to nodes in site B until the request is satisfied (i. e., file is finally found or does not exist in all nodes).

Figure 3 summarizes the circumstances when GMount performs better and worse than 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 client. However, for GMount, the latency of file lookup depends on how target file is close to the client. Therefore, GMount can benefit

from data access locality to outperform central metadata server approach. However, GMount can fall behind when target files are always in remote sites or do not exist since in this case a global search through many/every node is required.

2.3 Performance Improvements

Our evaluation on previous GMount implementation has pointed out two issues mainly related to I/O performance[2].

One problem is the limited data transfer rate over SSH channel. For this problem, we use a proxy-like approach to bypass SSH and conduct data transfer over raw sockets instead. This approach enables data transfer to benefit from TCP auto-tuning feature provided by most modern OS kernels[7].

The other problem is the I/O congestion since all I/O traffic is delivered indirectly over the overlay. For this problem, we use a file location piggybacking approach to pass the real file storage location to requesting client in initial file lookup reply. As a result, clients establish extra direct connections (separate from overlay) on demand to the nodes that hold the target data for I/O operations.

3. Evaluation

3.1 Experimental Environments

Our experiments use the InTrigger multi-cluster platform[8], 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 nodes have uniform installation and configurations of software: 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 4 sites for the evaluation. Figure 4 shows the configuration of these nodes, with the RTT (Round Trip Time) of network links between sites. Note that the RTT between nodes of same site is around 0.15ms.

The micro-benchmark used in experiments is ParaMark[9], which can issue parallel metadata or I/O requests with configurable access pattern from multiple clients to stress file system.

We compare GMount with Gfarm[1),10)], 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 harnessing local file system of computation nodes) and it has been deployed on the InTrigger platform for cross-site data

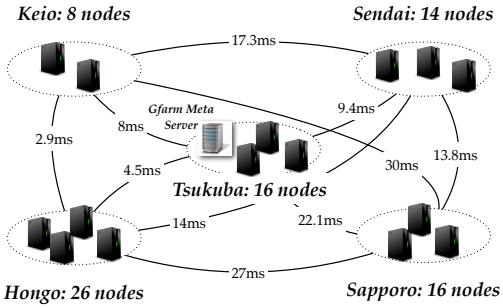

Fig. 4 Configuration of Experimental Environments

sharing purpose, it allows us to conduct an accurate and fair comparison. 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 performance. 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 *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 to servers that are located in the same cluster/LAN as the requesting client.

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 client). Then this sequence is applied to access data file 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 cases: (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 Figure 5, it reads that the metadata operations performance of Gfarm scales up to 10000ops/sec when the number of clients is 16 for most metadata operations except `rmdir`. Though *Tsukuba* site has only 16 nodes in

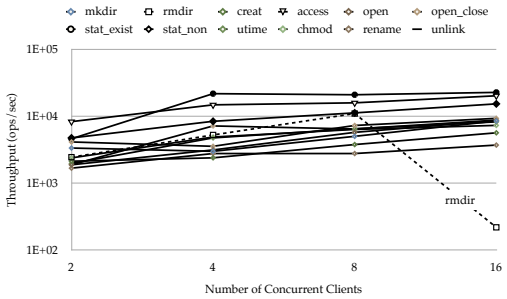

Fig. 5 Parallel Metadata Operations Performance of Gfarm in LAN ($r_{adj} = 1$)

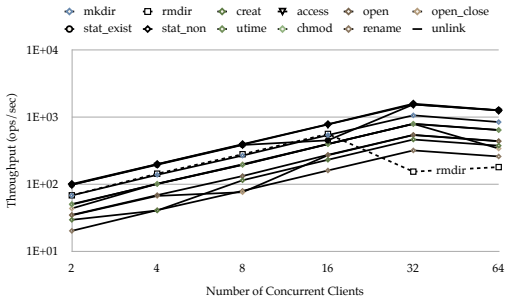

Fig. 6 Parallel Metadata Operations Performance of Gfarm in WAN ($r_{adj} = 1$)

this test, the metadata operations throughput does not scale significantly when the number of clients increases from 4 to 16.

In WAN, Gfarm achieves only 1600ops/sec peak performance for 32 clients in our experimental configuration. Similar results have been reported in another evaluation of Gfarm in InTrigger environment[1)], where `creat` achieves about 1800ops/sec for 32 clients and about 2400ops/sec for 64 clients. Though 2400ops/sec for 64 clients is higher than the performance we measured in our test, both results are in the same order of magnitude and the differences can be due to the selection of different clients. The high latency between clients and metadata servers results the overall low metadata operations performance of Gfarm in the wide-area environments.

Instead, as shown in Figure 7, the metadata operations performance of GMount scales up to an average of 70000ops/sec for 64 clients in WAN environment, this mainly attributes to the locality-aware metadata lookups. Note that our result is also comparable to the 100000ops/sec throughput of Ceph (using multiple metadata servers) for 128 distributed data servers in LAN[11)].

Figure 8 and Figure 9, respectively, show

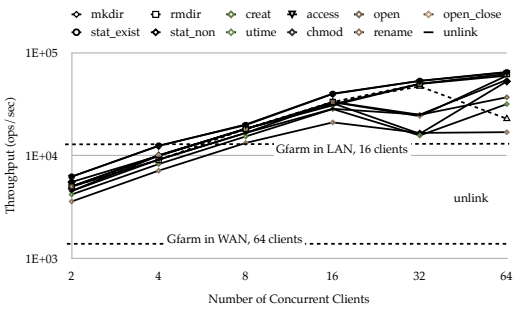

Fig. 7 Parallel Metadata Operations Performance of GMount in WAN ($r_{adj} = 1$)

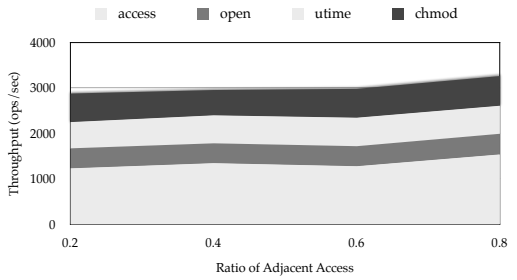

Fig. 8 Parallel Metadata Operations Performance of Gfarm in WAN ($0.2 < r_{adj} < 0.8$)

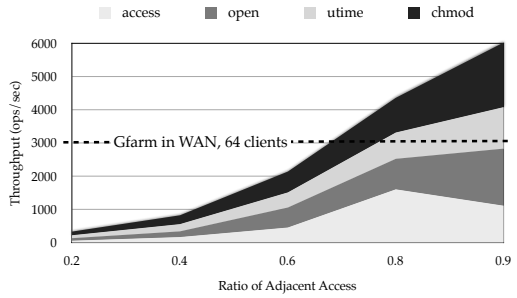

Fig. 9 Parallel Metadata Operations Performance of GMount in WAN ($0.2 < r_{adj} < 0.9$)

the performance results of Gfarm and GMount with 64 concurrent clients when there are references to remote files, or $r_{adj} \neq 1$. While the performance of Gfarm remains constant with the latency between clients and metadata servers, GMount is sensitive to data access locality of requests. As shown in Figure 9, GMount achieves an equivalent performance as Gfarm when $r_{adj} \approx 0.65$. GMount is about 5x slower than Gfarm when there is only 20% adjacent data accesses, because 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

Using the same configuration for metadata tests, we evaluate the parallel read and write

Fig. 10 Comparison of Parallel Read Performance

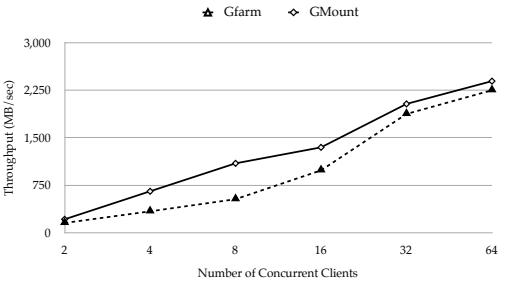

Fig. 11 Comparison of Parallel Write Performance

performance using 2GB file size and 128KB block request size for each client.

Results in Figure 10 and Figure 11 show that the performance of Gfarm and GMount both scales with the number of clients, while GMount has an overall 10%-20% better throughput than Gfarm. One possible overhead of Gfarm I/O is that data servers need to synchronize metadata updates to metadata server. Comparing to our I/O evaluation of previous GMount implementation[2], 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 applications to investigate the workflow execution performance by using GMount and Gfarm. We used GXP make[12], 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 is initially put in one node and distributed accordingly during the execution by file systems.

3.4.1 Event Recognition

The event recognition application[13] 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,

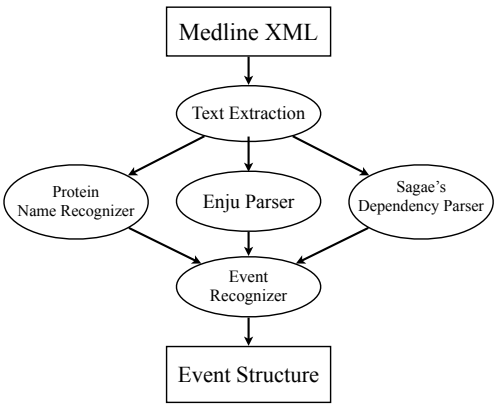

Fig. 12 Workflow DAG of Event Recognition

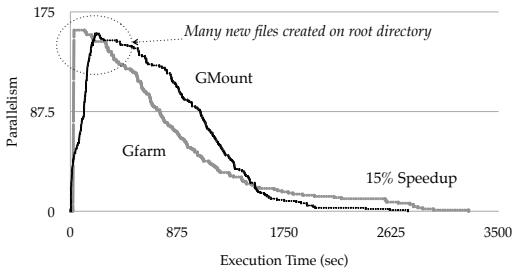

Fig. 13 Execution Profiles of Event Recognition

and a regulation of certain reactions. Its execution structure (workflow DAG) is shown in Figure 12. The input dataset consists of 158 input data (i. e., medline XML file), where each input data corresponds to a processing unit (the entire DAG chart) shown in Figure 12. We allow a maximum of 4 jobs to run concurrently on each node, resulting an overall maximum parallelism of 252.

Figure 13 shows the comparison of 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 159 by using Gfarm was reached at 28 second, and 156 parallelism by using GMount was reached at 198 second. Workflow running on Gfarm has faster initial scheduling performance is because many new files/directories are created under the work directory during the data 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.

To verify this finding, we extract and collect

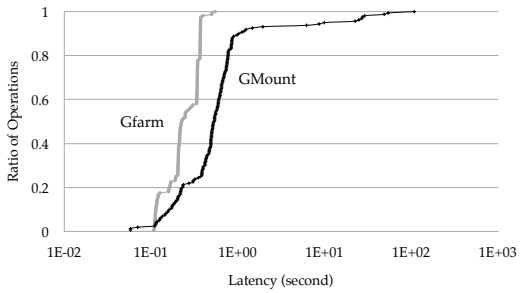

Fig. 14 Latency of File Creation in Root Directory

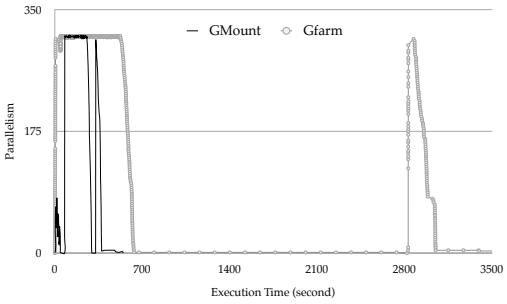

Fig. 15 Execution Profiles of Montage

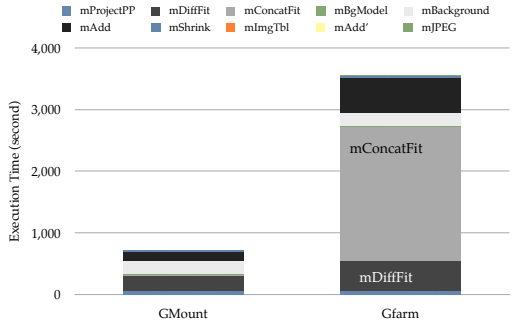

Fig. 16 Comparison of Execution Time of Phases

the execution time of those small jobs that create new files under the working directory by using GMount and Gfarm. The distribution of execution time of these jobs is presented in Figure 14. 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 due to concurrent creations on many nodes, resulting saturated data traffic in GMount overlay.

3.4.2 Montage

Another workflow is Montage astronomy scientific application[14], a popular benchmark that has been widely used in workflow studies[15] . The input data used in this experiment 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 set a maximum of 8 jobs to run concurrently on each node, resulting an overall maximum parallelism of 320.

Figure 15 shows the comparison of execution profiles, where the workflow finished in 3638 seconds when using Gfarm and 547 seconds when using GMount. Like event recognition workflow, GMount has lower performance at the initial phase `mProjectPP` because many files are created under the root directory.

Figure 16 shows the comparison of execution time of each phase. In one of major data process phases `mDiffFit`, GMount has a 2x speedup than Gfarm. Surprisingly, the largest difference comes from the phase `mConcatFit`, where 2000 seconds are cost in Gfarm but only 20 seconds are used in GMount. We found that `mConcatFit` is a significant data aggregation phase which collects (i. e., offset read from) small fraction of data from many files ($>$ 1000 files). In Gfarm, files opened for read are first replicated/cached on client node, leading heavy copy operations. In GMount, client only read required portion of data from each remote server over the network with specified access offset. Though the performance comparison of `mConcatFit` phase does not mainly related to data access locality, it shows that caching/replication have an adverse effect on performance in some cases.

4. Related Work

Most existing high-performance distributed file systems consist of dedicated file servers and are designed for the single-cluster environments, such as PVFS[16], Lustre[17], GPFS[18], GoogleFS[19], and HadoopFS[20]. To the best of our knowledge, they have not been widely deployed in multi-cluster environments except GPFS-WAN[21]. Experimental deployment and evaluation of Lustre wide-area environments have been studied[22),23], in which the complexity of deployment and low scalability and performance in wide-area have also been reported. Ceph[11] is an experimental peta-scale

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

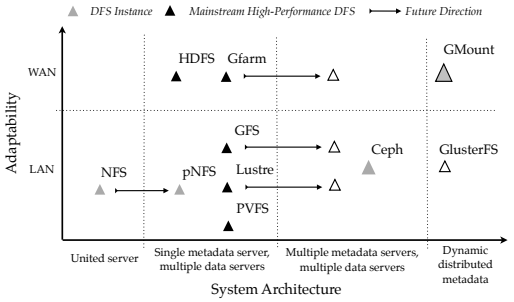

Fig. 17 Architectural Comparison of Existing Distributed File Systems

parallel file system, using a cluster of metadata servers to achieve scalable metadata operations. It also uses dedicated file servers. The effect of multiple metadata servers has not been studied in multi-clusters environments.

Figure 17 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 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 data footprint in application and the number of clients, single metadata server easily reaches its capacity and becomes the limitation for further performance improvement. Therefore, the *distributed management of metadata* is introduced, and one typical example of this design is Ceph. In addition, GFS shows its future plan of using distributed namespace servers[24]. Lustre and Gfarm also intend to implement multiple metadata servers[10),17)]. Though without many details, GlusterFS also gives a hint on using dynamic distributed metadata technique, similar as GMount, to achieve better scalability[25].

To comprehensively clarify the differences between GMount and other distributed file systems, we state that GMount should not be considered as a persistent distributed storage system and its usage scenario is different.

First, GMount is an *instantaneous* distributed file system that is supposed to be constructed on demand and destructed after the usage. It is a complementary distributed file system for data sharing practices (i. e., execute data-intensive applications among cross-domain resources in the wide area) where traditional distributed file systems can not be straightforwardly applied or work well as expected.

Second, as tradeoffs to achieve better performance when there are more data access locality and instantaneous deployment, GMount adopts a weaken cache consistency model and sacrifices the fault tolerance in current implementation. GMount does not have its own cache subsystem or create data replicates for I/O or fault tolerance. Data are manipulated directly on the target data stored in remote servers over the network. Since GMount directly harnesses the local file system to store user data, the data availability of GMount is dedicated to local filesystems of each host.

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 can 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 system for global data sharing on arbitrary resource, GMount is practically useful for general users to conduct their data-intensive distributed computing practice. It also shows a way of building the distributed file system with a small implementation.

The major future work includes a design and implementation of sophisticated metadata management to improve the metadata lookup performance by reducing the overhead of global searching when file creation and data access locality is not significant. There is also a plan to implement a simple interface to allow workflow management systems to query file locations for file affinity scheduling.

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

Acknowledgements

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) Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm grid file system. *New Generation Computing*, 28:257–275, 2010.
2) Nan Dun, Kenjiro Taura, and Akinori Yonezawa. GMount: An ad hoc and locality-aware distributed file system by using SSH and FUSE. In *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid*, CCGrid '09, pages 188–195, May 2009.
3) Miklos Szeredi. FUSE: Filesystem in userspace. Online available at `http://fuse.sourceforge.net/`.
4) Nan Dun. SSHFS-MUX. Online available at `http://sshfsmux.googlecode.com/`.
5) Charles P. Wright, Jay Dave, Puja Gupta, Harikesavan Krishnan, David P. Quigley, Erez Zadok, and Mohammad Nayyer Zubair. Versatility and Unix semantics in namespace unification. *ACM Transactions on Storage*, 2:74–105, February 2006.
6) Kenjiro Taura. GXP: An interactive shell for the grid environment. In *Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, IWIA '04, pages 59–67, Charlotte, NC, USA, April 2004.
7) Enabling high performance data transfers. Online available at `http://www.psc.edu/networking/projects/tcptune/`.
8) InTrigger multi-cluster platform. Online available at `http://www.intrigger.jp/`.
9) Nan Dun. ParaMark: Parallel filesystem benchmark. Online available at `http://paramark.googlecode.com/`.
10) Gfarm. Online available at `http://datafarm.apgrid.org/`.
11) Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 1–7, 2006.
12) Kenjiro Taura, Takuya Matsuzaki, Makoto Miwa, Yoshikazu Kamoshida, Daisaku Yokoyama, Nan Dun, Takeshi Shibata, Sungjun Choi, and Jun'ichi Tsujii. Design and implementation of GXP Make – a workflow system based on make. In *Proc. IEEE e-Science 2010 Conference*, e-Science '10, December 2010.
13) Makoto Miwa, Rune Sætre, Jin-Dong Kim, and Jun'ichi Tsujii. Event extraction with complex event classification using rich features. *Journal of Bioinformatics and Computational Biology*, 8(1):131–146, February 2010.
14) Joseph C. Jacob, Daniel S. Katz, G. Bruce Berriman, John C. Good, Anastasia C. Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas A. Prince, and Roy Williams. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4:73–87, July 2009.
15) Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.
16) Parallel Virtual File System. Online available at `http://www.pvfs.org/`.
17) Lustre file system. Online available at `http://www.lustre.org/`.
18) Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. Conference on File and Storage Technologies*, pages 231–244, January 2002.
19) Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proc. The 9th ACM Symposium on Operating Systems Priciples*, SOSP '03, pages 29–43, New York, NY, USA, October 2003.
20) Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies*, MSST '10, May 2005.
21) Phil Andrews, Patricia Kovatch, and Christopher Jordan. Massive high-performance global file systems for grid computing. In *Proc. the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, Washington, DC, USA, 2005. IEEE Computer Society.
22) Jason Cope, Michael Oberg, Henry M. Tufo, and Matthew Woitaszek. Shared parallel file systems in heterogeneous linux multicluster environments. In *Proc. The 6th LCI International Conference on Linux Clusters*, April 2005.
23) StephenC. Simms, GregoryG. Pike, and Doug Balog. Wide area filesystem performance using lustre on the TeraGrid. In *Proc. The TeraGrid 2007 Conference*, June 2007.
24) Kirk McKusick and Sean Quinlan. GFS: Evolution on fast forward. *Queue*, 53(3):42–49, March 2010.
25) GlusterFS. Online available at `http://www.gluster.org/`.