# Distributed Data Provenance for Large-Scale Data-Intensive Computing

Dongfang Zhao*, Chen Shou*, Tanu Malik†‡, Ioan Raicu*‡
*Department of Computer Science, Illinois Institute of Technology
†Computation Institute, The University of Chicago
‡Math and Computer Science Division, Argonne National Laboratory

***Abstract*—It has become increasingly important to capture and understand the origins and derivation of data (its provenance). A key issue in evaluating the feasibility of data provenance is its performance, overheads, and scalability. In this paper, we explore the feasibility of a general metadata storage and management layer for parallel file systems, in which metadata includes both file operations and provenance metadata. We experimentally investigate the design optimality—whether provenance metadata should be loosely-coupled or tightly integrated with a file metadata storage systems. We consider two systems that have applied similar distributed concepts to metadata management, but focusing singularly on kind of metadata: (i) FusionFS, which implements a distributed file metadata management based on distributed hash tables, and (ii) SPADE, which uses a graph database to store audited provenance data and provides distributed module for querying provenance. Our results on a 32-node cluster show that FusionFS+SPADE is a promising prototype with negligible provenance overhead and has promise to scale to petascale and beyond. Furthermore, FusionFS with its own storage layer for provenance capture is able to scale up to 1K nodes on BlueGene/P supercomputer.**

## I. Introduction

Scientific advancement and discovery critically depends upon being able to extract knowledge from extremely large data sets, produced either experimentally or computationally. In experimental fields such as high-energy physics datasets are expected to grow by six orders of magnitude [1]. In computational fields such as fusion science data will be output at 2 gigabytes/second per core or 2 petabytes/second of checkpoint data every 10 minutes [1]. This amounts to an unprecedented I/O rate of 3.5 terabytes/second. To extract knowledge from extremely large datasets in a scalable way, architectural changes to HPC systems are increasingly being proposed—changes that either reduce simulation output data [2, 3] or optimize the current flop to I/O imbalance [4, 5]. Many-Task Computing (MTC) [6, 7] was recently proposed as a new paradigm to bridge the gap between HPC and high-throughput computing.

A primary architectural change is a change in the design of the storage layer, which is currently segregated from compute resources. Storage is increasingly being placed close to compute nodes in order to help manage large-scale I/O volume and data movement [4, 8–10], especially for efficient checkpointing at extreme scale [11]. This change in the storage layer has a significant resulting advantage—it enables simulation output data to be stored with the provenance metadata so that analysis can be easily verified, validated as well as retraced over time steps even after the simulation has finished.

978-1-4799-0898-1/13/$31.00 

While this architectural change is being deemed necessary to provide the much needed scalability advantage of concurrency and throughput, it cannot be achieved without providing an efficient storage layer for conducting metadata operations [12]. The centralized metadata repository in parallel file systems has shown to be inefficient at large scale for conducting metadata operations, growing for instance from tens of milliseconds on a single node (four-cores), to tens of seconds at 16K-core scales [12, 13]. Similarly, auditing and querying of provenance metadata in a centralized fashion has shown poor performance over distributed architectures [14].

In this paper, we explore the feasibility of a general metadata storage and management layer for parallel file systems, in which metadata includes both file operations and provenance metadata. In particular we experimentally investigate the design optimality—whether provenance metadata should be loosely-coupled or tightly integrated with a file metadata storage systems. To conduct this experimental evaluation, we consider two systems that have applied similar distributed concepts to metadata management, but focusing singularly on kind of metadata: (i) FusionFS [15], which implements a distributed file metadata management based on distributed hash tables, and (ii) SPADE [16], which uses a graph database to store audited provenance data and provides distributed module for querying provenance.

Both FusionFS and SPADE are good choices for investigating the metadata storage design problem since both systems have similar manifestation of distributed concepts towards storing their individual metadata: (1) FusionFS provides a POSIX interface which makes a perfect corresponding for SPADE user-level file system (FUSE-based) provenance collection; (2) both systems work in a decentralized way thus actively exploiting the resources at each node.

This paper first introduces the `SPADE+FusionFS` version of provenance-aware distributed file system, that aims to offer excellent scalability while retaining the provenance overhead negligible in traditional clusters. Some preliminary results of `SPADE+FusionFS` have been published in [17]. This paper then investigates using Zero-hop Distributed Hashtable (ZHT) [18] as the underlying storage system for provenance. ZHT is currently used to store file metadata in FusionFS and provides the following features that makes it a desirable choice to store provenance: (1) excellent storage load balancing; (2) light-weighted and fast; (3) excellent scalability; (4) be able to provide a global view of provenance that aims to

provide provenance capture and management in petascale and exascale. We term the ZHT-backed provenance system as `FusionProv`.

Our results on a 32-node cluster show that FusionFS+SPADE is a promising prototype with negligible provenance overhead and has promise to scale to petascale and beyond. FusionFS on its own, has shown to scale to 1K-nodes [15], and its design has been architected to scale to 1M-nodes, i.e. exascale. `FusionProv` has a similar performance on provenance capture compared with `SPADE+FusionFS` on traditional Linux clusters while beating SPADE in query executing time. `FusionProv` is able to scale up to 1K nodes on BlueGene/P supercomputer. Its light-weight implementation in C++ and global views account for its high performance.

In summary, this papers has the following contributions:

- Design and implement FusionProv, a distributed provenance-aware file system,
- Propose a hybrid coarse/fine grained approach to minimize the amount of provenance data captured while maintaining provenance granularity and detail
- Evaluate the performance at up to 1K-node scales (potentially the largest data provenance evaluation to date)

The remainder of this paper is organized as follows. We review some related work in Section 2. Section 3 describes the building blocks of the two provenance systems. We present the design and implementation of FusionFS+SPADE and FusionProv in Section 4. Section 5 evaluates both systems. We conclude this paper and discusses the future work in Section 6.

## II. Related Work

As distributed systems become more ubiquitous and complex, there is a growing emphasis on the need for tracking provenance metadata along with file system metadata. A good review is presented in [19]. Many Grid systems like Chimera [20] and the Provenance-Aware Service Oriented Architecture (PASOA) [21] provide provenance tracking mechanisms for various applications. However these systems are very domain specific and do not capture provenance at the file system level. The Distributed Provenance Aware Storage System (DPASS) tracks the provenance of files in a distributed file system by intercepting file system operations and sending this information via a netlink socket to user level daemon that collects provenance in a database server [22]. The provenance is however, collected in a centralized fashion, which is a poor design choice for distributed file systems meant for extreme scales. Similarly in efficient retrieval of files, provenance is collected centrally [23].

PASS describes global naming, indexing, and querying in the context of sensor data [24]. PA-NFS [25] enhances NFS to record provenance in local area networks but does not consider distributed naming explicitly. SPADE [16] addresses the issue by using storage identifiers for provenance vertices that are unique to a host and requiring distributed provenance queries to disambiguate vertices by referring to them by the host on which the vertex was generated as well as the identifier local to that host.

Several storage systems have been considered for storing provenance. ExSPAN [26] extends traditional relational models for storing and querying provenance metadata. SPADE supports both graph and relational database storage and querying. PASS has explored the use of clouds [24]. Provbase uses Hbase to store and query scientific workflow provenance [27]. Further compressing provenance [26], indexing [14] and optimization techniques [28] have also been considered. However, none of these systems have been tested for exascale architectures. To give adequate merit to the previous designs we have integrated FusionFS with SPADE as well as considered FusionFS's internal storage system for storing audited provenance.

## III. Building Blocks

### A. FusionFS

FusionFS is a new distributed file system designed from the ground up for high scalability (1K-nodes) while achieving significantly higher I/O performance (1.02 TB/sec). FusionFS achieves these levels of scalability and performance through complete decentralization, and the co-location of storage and compute resources. It supports POSIX-like interfaces important for ease of adoption and backwards compatibility with legacy applications. It is made reliable through data replication, and it supports both strong and weak consistency semantics. FusionFS has been deployed on a variety of testbeds, ranging from a 32-node (256-cores) Linux cluster, to a 96-VM virtual cluster on the Amazon EC2 cloud, to a 1K-node (4K-cores) IBM BlueGene/P supercomputer with promising results, when compared to other leading storage systems such as GPFS, PVFS, HDFS, and S3.

A high-level structure of FusionFS is illustrated in Figure 1. Each compute node plays the same role in FusionFS and everything is completely decentralized. These compute nodes are normally interconnected by some high performance network (e.g. 3-D torus InfiniBand in IBM Blue Gene/P). The high bandwidth of the node-to-node communication is crucial to the success of FusionFS.

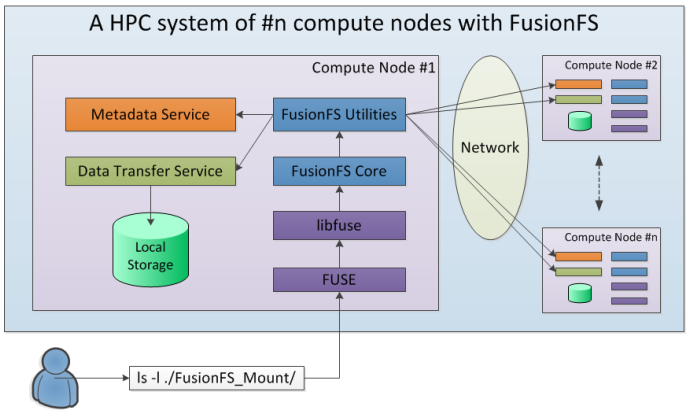


Fig. 1. Architectural overview of FusionFS

Components of each node are briefly explained in the following:

1) **FUSE:** FUSE is the Linux kernel module that monitors any I/O requests made to FusionFS. The caught file request is then transmitted to the libfuse library. FUSE supports POSIX interfaces, so that FusionFS is backwards compatible with commonly found parallel file systems and local file systems (e.g. EXT4).
2) **libfuse:** libfuse is a user-level library that interprets the incoming POSIX-compliant file requests into FusionFS implementation. After that it releases the control to the FusionFS Core module. Both FUSE and libfuse are parts of the FUSE framework [29], that has been used in many research filesystem prototypes.
3) **FusionFS Core:** This module implements all the FUSE interfaces to manipulate POSIX file operations. For example, when a user opens a file, the $open()$ function will be triggered and possibly need some auxiliary functions from FusionFS Utilities.
4) **FusionFS Utilities:** This module provides miscellaneous utilities supporting local FusionFS Core module and local services, as well as communication to remote compute nodes. For example, metadata management is implemented in FusionFS Utilities on top of the metadata service.
5) **Metadata Service:** It is a service dedicated for metadata manipulations. The infrastructure is essentially a lightweight and efficient distributed hash table (i.e. ZHT) dispersed over compute nodes. A distributed metadata service avoids the potential bottleneck or single point of failure commonly found in metadata management of traditional parallel or distributed filesystems.
6) **Data Transfer Service:** It is a service that handles data transfer. The data management and metadata management are completely decoupled, allowing different strategies for each. For example, metadata should be randomly distributed on many nodes to achieve good load balance, but data should be located on nodes in proximity of the application reading or writing the data, maximizing its locality. This independence makes FusionFS scalable on both metadata throughput and I/O throughput.
7) **Local Storage:** We assume there is a high performance persistent storage (e.g. SSD) attached to each compute node. It helps exploit data locality, and is particularly useful for data-intensive distributed systems [30].

The software stack of FusionFS is shown in Figure 2. Two services (metadata, data transfer) are on top of the stack, that are supported by FusionFS Core and FusionFS Utilities interacting with the kernel FUSE module.

### B. ZHT

ZHT [18] (Zero-hop distributed Hash Table) was originally designed as a general-purpose distributed hash table (i.e. key-value store) for HPC. Key-value store was shown to be viable at extreme scales [31]. We extended ZHT in two directions:

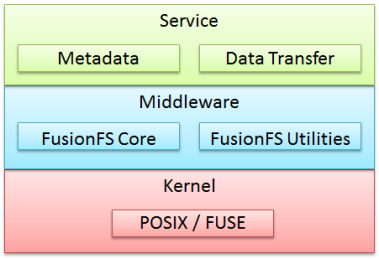


Fig. 2. FusionFS software stack

1) it serves as the underlying storage for metadata of FusionFS [15] filesystem, and 2) it is tuned to be the underlying infrastructure of the IStore key-value storage [32]. We would also leverage ZHT to manage the HyCache [33] heterogeneous storage system. ZHT has scaled up to 32K cores on IBM Blue Gene/P [34] supercomputer and 96 Amazon EC2 [35] instances.

As shown in Figure 3, ZHT has a similar ring-shaped look as the traditional DHT [9]. The node IDs in ZHT can be randomly distributed across the network. The correlation between different nodes is computed with some logistic information like IP address, for example. The hash function maps a string to an ID that can be retrieved by a $lookup(k)$ operation at a later point.

There are multiple choices of distributed hash table (DHT) available, e.g. Memcached [36] and Dynamo [37] etc. We chose ZHT as the underlying distributed hash table because it has some features that are critical to the success of FusionFS. As summarized in Table I, ZHT has many advantages, such as being implemented in C/C++, having the lowest routing time, and supporting both persistent hashing and dynamic membership.

TABLE I
COMPARISONS BETWEEN DIFFERENT DHT IMPLEMENTATIONS

| | ZHT | Memcached | Dynamo |
|---|---|---|---|
| Impl. Language | C/C++ | C | Java |
| Routing Time | 0 - 2 | 2 | 0 - $\log N$ |
| Persistence | Yes | No | Yes |
| Dynamic Member | Yes | No | Yes |

### C. FDT

To migrate data across different nodes, we need a data transfer service that is efficient, reliable and light-weight. User Datagram Protocol (UDP) is efficient in transferring data, but is an unreliable protocol. Transmission Control Protocol (TCP), on the other hand, is reliable but has a relatively lower efficiency. Ideally, a hybrid UDP/TCP protocol might be best; essentially a protocol that is both reliable and efficient.

We have developed our own data transfer service called FDT (Fast Data Transfer) with APIs provided by UDP-based Data Transfer (UDT) [38], which is a reliable UDP-based application level data transport protocol for distributed

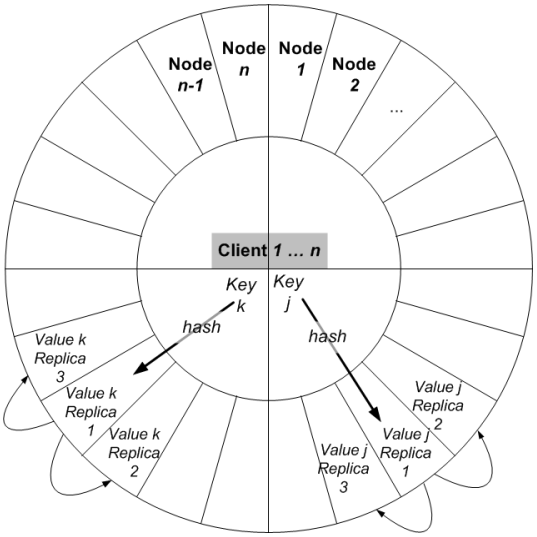


Fig. 3. ZHT architecture: namespace, hash function and replications

data-intensive applications. UDT adds its own reliability and congestion control on top of UDP which thus offers potentially higher speed than TCP under certain conditions.

### D. SPADE

SPADE is a software infrastructure for data provenance collection, management, and analysis. Different operating system level *reporters* facilitate provenance collection. The underlying data model is graph-based, consisting of vertices and directed edges, each of which can be labeled with an arbitrary number of annotations (in the form of key-value pairs). These annotations can be used to embed the domain-specific semantics of the provenance. The SPADE system decouples the production, storage, and utilization of provenance metadata, as illustrated in Figure 4. At its core is a provenance kernel that mediates between the producers and consumers of provenance information, and handles the persistent storage of records. The kernel handles buffering, filtering, and multiplexing incoming metadata from multiple provenance sources. It can be configured to commit the elements to multiple databases, and responds to concurrent queries from local and remote clients.

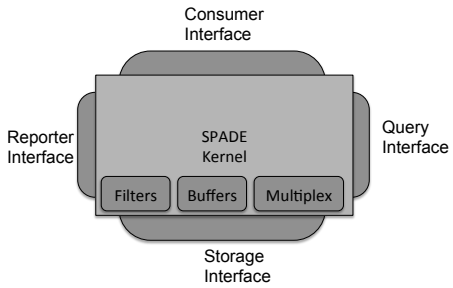


Fig. 4. The SPADE architecture

## IV. Design and Implementation

### A. SPADE+FusionFS: SPADE Extension with FusionFS

*1) Design:* The architecture of SPADE+FusionFS integration is shown in Figure 5. Each node has two services installed: FusionFS service and SPADE service. One service type can only communicate to the other type on the local node. That is, a SPADE service only communicates with its local FusionFS service, and vice versa. For services of the same type (e.g. FusionFS ⇔ FusionFS, SPADE ⇔ SPADE), they are free to talk to others remotely.

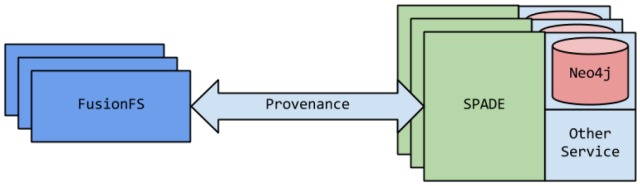


Fig. 5. FusionFS+SPADE architecture overview

In order to make the collected provenance compliant to the Open Provenance Model (OPM), when there is a network transmission, SPADE creates a "dummy" FusionFS process vertex to connect two artifacts: a file vertex and a network vertex. We call it a "dummy" process because clients do not need to be concerned with this process when querying provenance; it is just a symbol to indicate the network transmission is triggered by FusionFS in OPM. Figure 6 shows how a network transmission is represented.

*2) Implementation:* The key challenge of the proposed work is how to seamlessly integrate SPADE and FusionFS. All communication between these two services is implemented with TCP. Asynchronous communication is not used because of the short life cycle of some processes. SPADE collects parts of the process information based on system files under directory /proc/pid. If a process starts and terminates too fast for SPADE to catch, there would be provenance loss. Therefore it is critical to keep synchronous communication between SPADE and FusionFS, at least while the two systems are completely decoupled. We hope to address this in future work with a tighter integration between FusionFS and SPADE.

Most communication between SPADE and FusionFS consists of simple operation bindings. For example, FusionFS write operation invokes SPADE to collect write provenance for this operation. However, as a distributed file system, FusionFS sometimes needs to migrate files between nodes. The original network provenance collection in SPADE is not optimized for FusionFS. So we make some customization to the network provenance collection to fully hide unnecessary provenance data outside FusionFS.

*3) File-Level vs. Block-Level:* One common practice in file manipulations is to split (large) files into blocks to improve the space efficiency and responsive time. However, for the purpose of provenance, it is less interesting to keep track of file traces at the block level: in most cases, a file-level provenance would

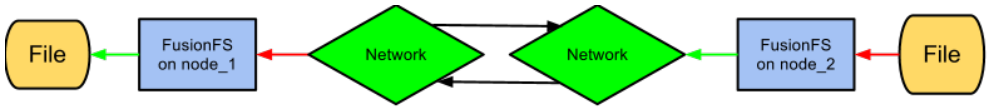


Fig. 6. Network Transmission

suffice. We have implemented both the file-level and the block-level provenance tracings, namely, the fine-grained provenance and the coarse-grained provenance.

### B. FusionProv: Distributed Provenance with ZHT and FusionFS

*1) Design:* Figure 7 illustrates how we integrate FusoinFS and ZHT to support distributed provenance capture at the file system level. Provenance is firstly generated in the FUSE layer in FusionFS, and then is cached in the local provenance buffer. And at a certain point (e.g. when the file is closed), the cached provenance will be persisted into ZHT. Users can do query on any node of the system using a ZHT client.

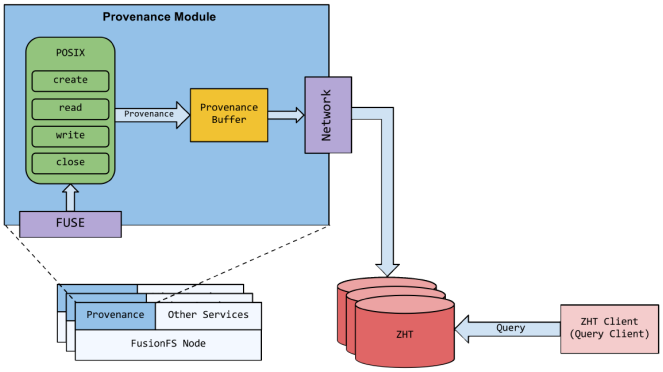


Fig. 7. FusionFS+ZHT architecture overview

*2) Implementation:* Table II shows what is captured for the graph vertex in the distributed provenance store. Basically there are two different vertex types being tracked of: file and process. In other words, we are interested in which file(s) have been touched by which process(es). And we maintain a linked list for the tree topology in ZHT.

We provide a set of APIs to allow users plug their own implementations for the provenance they are interested in. Some commonly used APIs are listed in Table III. Note that for file creation, there is no need to save the provenance in the local buffers because it only touches the metadata (rather than the file/process). Therefore this information is directly stored in the underlying metadata storage (i.e. ZHT).

We implement a light-weight command-line tool that end users can use to query the provenance, in the following syntax:

```
query vertex [filename] [file version]
  [ancestors -- descendants] [depth]
```

For example, with a following workflow: a file (origin_file) was created by a touch process on host 12.0.0.1, and later was copied by multiple processes on multiple nodes (12.0.0.2 to 12.4.0.16). The query on the descendants of the touch process (vertex) would generate provenance graph showed in Figure 8.

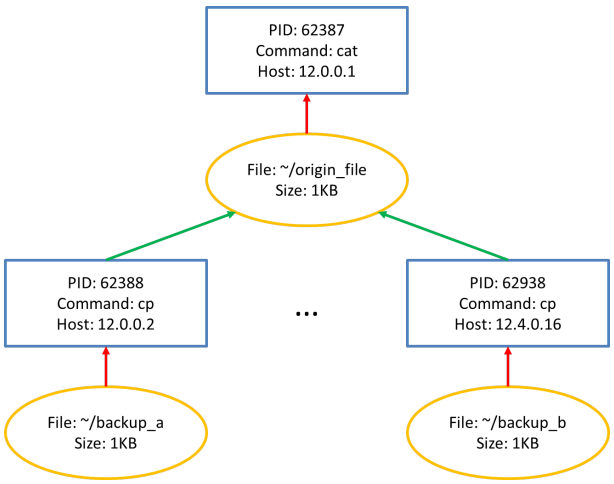


Fig. 8. An example query tree in distributed provenance systems

## V. Evaluation

We have deployed the distributed provenance-aware file system on 1K-node IBM BlueGene/P supercomputer Intrepid [34]. We also evaluated both the distributed and SPADE-extended systems on a 32-node cluster, where each node has two Quad-Core AMD Opteron 2.3GHz processors with 8GB memory. All nodes are interconnected by 1Gbps Ethernet. All experiments are repeated at least 3 times to obtain stable results (i.e. within 5% difference).

### A. SPADE + FusionFS

*1) Single-Node Throughput:* We first measured the performance of provenance collection within FusionFS on a single node. A client reads/writes a 100MB file from/to FusionFS. We compare the performance between fine-grained and coarse-grained provenance collection with different block sizes. The benchmark we used is IOZone [39], which is carefully tuned to avoid operating system cache.

Figure 9 and Figure 10 show that a fine-grained provenance collection introduces a high overhead. Even though a larger block size could reduce the overhead to some degree, the number is still significantly high (i.e. around 75%), compared to coarse-grained provenance (i.e. less than 5%). This is expected since a bigger I/O block size results in fewer I/O runs, which further involves less time to collect provenance (SPADE

TABLE II
ATTRIBUTES OF GRAPH VERTEXES IN DISTRIBUTED PROVENANCE CAPTURE

| Vertex Type | Attributes |
|---|---|
| File | [File path/name] [File version] [File size] |
| Process | [Process host] [Process name] [Process command line] |

TABLE III
SOME EXAMPLE APIS AVAILABLE FOR PROVENANCE CAPTURE

| FusionFS Operation | Provenance API | Description |
|---|---|---|
| fusion_read() | prov_read() | Save the reading process and the file being read into the local buffer |
| fusion_write() | prov_write() | Save the writing process and the file being written into the local buffer |
| fusion_create() | prov_create() | Directly save the file and process info into ZHT |
| fusion_flush() | prov_push() | Summarize the info stored in the local buffer, and save the summary into ZHT |

spends on average 2.5 ms for each provenance recording, which corresponds to a single I/O run in the fine-grained provenance collection).

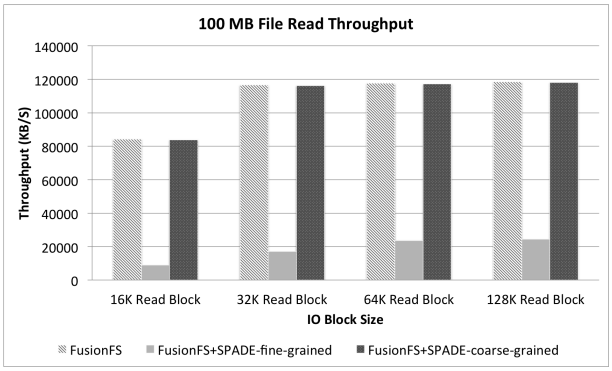


Fig. 9. Read Throughput

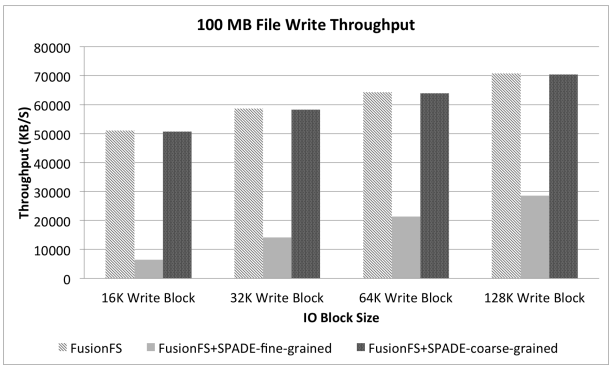


Fig. 10. Write Throughput

*2) Multi-Node Throughput:* In the 32-node cluster, multiple clients read/write distinct files from/to FusionFS. The file size is set to 100MB and the I/O block size is set to 128KB.

In Figure 11, a coarse-grained provenance collection shows a much better performance than the fine-grained counterpart (consistent with the single-node benchmark results). Both fine-grained and coarse-grained provenance show excellent scalability with linear increase in performance. This can be explained by two facts: (1) SPADE only collects provenance of the local node, and (2) FusionFS scales linearly with respect to the number of nodes by getting high data locality in the data access pattern evaluated. We have evaluated FusionFS (without SPADE) at scales of up to 1K nodes on a IBM BlueGene/P supercomputer with similar excellent results. We will conduct larger scale experiments of FusionFS+SPADE in future work.

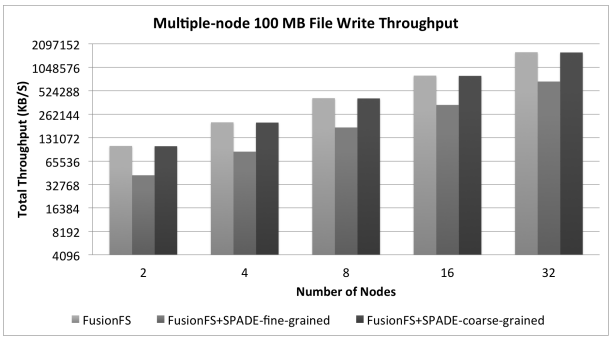


Fig. 11. Multiple-Node 100MB Write Throughtput

*3) Query time:* We are interested in the query time of the provenance of a particular file that has been read by multiple remote nodes. This write-once-read-many is a very frequent pattern in the context of a distributed system. The query is shown in the following format:

$$\texttt{query lineage descendants } vertex-id \texttt{ 100 null filename:} test.file.name$$

Since SPADE (with version) does not support executing sub-query in parallel, the total query time increases as it scales up. However, according to Figure 12, with different scales from 2 to 32 nodes, the average per-node query time is about constant, indicating that adding more nodes will not put more burden to the provenance system. This is expected, since the underlying FusionFS has an excellent scalability and SPADE on each node adds negligible overheads locally.

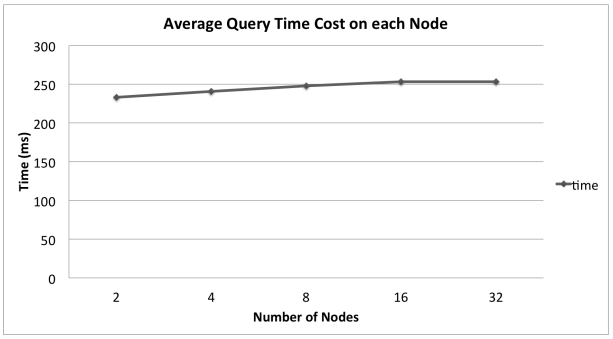


Fig. 12. Query Time Cost

### B. Distributed Provenance Capture and Query

*1) Provenance capture:* We compare the throughput of the distributed provenance capture to the `SPADE+FusionFS` implementation in Figure 13. The ZHT-based throughput is comparable to both the pure FusionFS and the coarse-grained `SPADE+FusionFS` implementations. This result suggests that, even though there is network overhead involved in distributed provenance capture, the cost is about negligible.

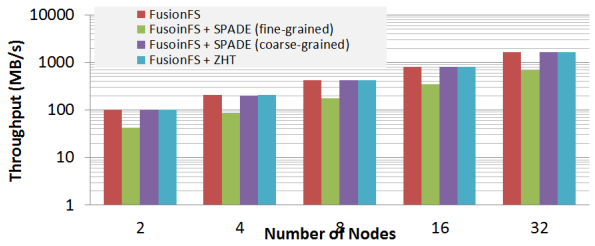


Fig. 13. Throughput of different implementations of provenance capture

*2) Provenance Query:* Similarly to throughput, we also compare the query time of different implementations. Figure 14 shows that even on one single node, the ZHT-based implementation is much faster than SPADE (0.35ms vs. 5ms). At 32-node scale, the gap is even larger result in 100X difference (108ms vs. 11625ms).

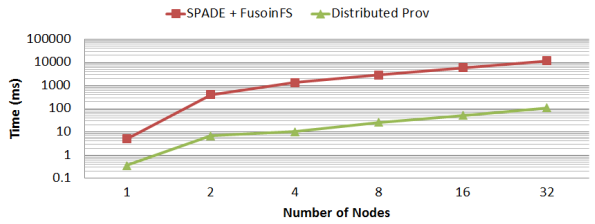


Fig. 14. Query time of different provenance system

*3) Scalability:* We have scaled the distributed provenance system up to 1K-node on IBM BlueGene/P. Figure 15 shows that the provenance overhead is relative small even on 1K nodes (14%). Similarly, we report the query time and overhead on the same workload at large scale (i.e. 1K nodes) in Figure 16, which shows that the overhead at 1K-nodes is about 18%.

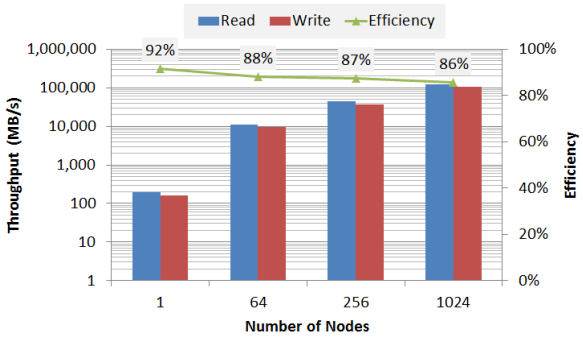


Fig. 15. Throughput on BlueGene/P

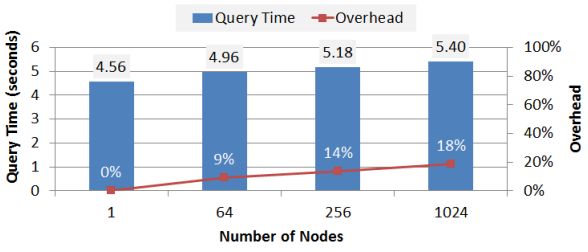


Fig. 16. Query Time on BlueGene/P

## VI. Conclusion and Future Work

This paper explores the feasibility of a general metadata storage and management layer for parallel file systems, in which metadata includes both file operations and provenance metadata. Two systems are investigated (1) FusionFS, which implements a distributed file metadata management based on distributed hash tables, and (2) SPADE, which uses a graph database to store audited provenance data and provides distributed module for querying provenance. Our results on a 32-node cluster show that FusionFS+SPADE is a promising prototype with negligible provenance overhead and has promise to scale to petascale and beyond. Furthermore, FusionFS with its own storage layer for provenance capture is able to scale up to 1K nodes on BlueGene/P supercomputer. As for the future work, we plan to integrate the Swift parallel programming system [40] to deploy real scientific applications [41, 42] on FusionFS+SPADE and FusionProv, as well as continue to scale FusionFS/FusionProv towards petascale levels.

## Acknowledgement

This work was supported by the National Science Foundation under grant OCI-1054974, and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of

the U.S. Department of Energy under contract DE-AC02-06CH11357. The authors are grateful to Xian-He Sun for providing the access to the Linux cluster.

## REFERENCES


[1] P. A. Freeman, *et al.*, "Cyberinfrastructure for science and engineering: Promises and challenges," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 682–691, 2005.

[2] K.-L. Ma, *et al.*, "In-situ processing and visualization for ultrascale simulations," in *Journal of Physics: Conference Series*, vol. 78, p. 012043, IOP Publishing, 2007.

[3] K.-L. Ma, "In situ visualization at extreme scale: Challenges and opportunities," *Computer Graphics and Applications, IEEE*, vol. 29, no. 6, pp. 14–19, 2009.

[4] S. Ghemawat, *et al.*, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, 2003.

[5] N. Ali, *et al.*, "Scalable i/o forwarding framework for high-performance computing systems," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[6] I. Raicu, *et al.*, "Middleware support for many-task computing," *Cluster Computing*, vol. 13, Sept. 2010.

[7] I. Raicu, "Many-task computing: Bridging the gap between high-throughput computing and high-performance computing," Doctoral dissertation, The University of Chicago, 2009.

[8] K. Shvachko, *et al.*, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, 2010.

[9] I. Stoica, *et al.*, "Chord: Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM)*, 2001.

[10] I. Raicu, *et al.*, "Making a case for distributed file systems at exascale," in *Proceedings of the third international workshop on Large-scale system and application performance*, LSAP '11, 2011.

[11] D. Zhao, *et al.*, "Exploring reliability of exascale systems through simulations," in *Proceedings of the High Performance Computing Symposium*, HPC '13, 2013.

[12] I. Raicu, *et al.*, "Toward loosely coupled programming on petascale systems," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, 2008.

[13] Z. Zhang, *et al.*, "Design and evaluation of a collective i/o model for loosely- coupled petascale programming," IEEE MTAGS, 2008.

[14] T. Malik, *et al.*, "Sketching distributed data provenance," in *Data Provenance and Data Management in eScience*, pp. 85–107, 2013.

[15] D. Zhao *et al.*, "Distributed File Systems for Exascale Computing," ACM/IEEE Supercomputing (Doctoral Showcase), 2012.

[16] A. Gehani *et al.*, "SPADE: Support for Provenance Auditing in Distributed Environments," ACM/USENIX Middleware, pp. 101–120, 2012.

[17] C. Shou, *et al.*, "Towards a provenance-aware distributed filesystem," in *5th Workshop on the Theory and Practice of Provenance*, 2013.

[18] T. Li, *et al.*, "ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table," in *IEEE International Parallel & Distributed Processing Symposium*, IEEE IPDPS '13, 2013.

[19] K.-K. Muniswamy-Reddy, "Foundations for provenance-aware systems," Doctoral dissertation, Harvard University, 2010.

[20] I. T. Foster, *et al.*, "The virtual data grid: A new model and architecture for data-intensive collaboration," in *CIDR'03*, pp. – 1–1, 2003.

[21] Provenance aware service oriented architecture, "http://twiki.pasoa.ecs.soton.ac.uk/bin/view/ pasoa/webhome."

[22] A. Parker-Wood, *et al.*, "Making sense of file systems through provenance and rich metadata," Tech. Rep. UCSC-SSRC-12-01, University of California, Santa Cruz, Mar. 2012.

[23] K.-K. Muniswamy-Reddy, *et al.*, "Provenance-aware storage systems," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, 2006.

[24] K.-K. Muniswamy-Reddy, *et al.*, "Making a cloud provenance-aware," in *1st Workshop on the Theory and Practice of Provenance*, 2009.

[25] K.-K. Muniswamy-Reddy, *et al.*, "Layering in provenance systems," in *Proceedings of the 2009 USENIX Annual Technical Conference*, 2009.

[26] W. Zhou, *et al.*, "Efficient querying and maintenance of network provenance at internet-scale," in *Proceedings of the 2010 international conference on Management of data*, pp. 615–626, ACM, 2010.

[27] J. Abraham, *et al.*, "Distributed storage and querying techniques for a semantic web of scientific workflow provenance," in *Services Computing (SCC), 2010 IEEE International Conference on*, pp. 178–185, IEEE, 2010.

[28] T. Heinis *et al.*, "Efficient lineage tracking for scientific workflows," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1007–1018, ACM, 2008.

[29] FUSE Project, "http://fuse.sourceforge.net."

[30] I. Raicu, *et al.*, "The quest for scalable support of data-intensive workloads in distributed systems," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, 2009.

[31] K. Wang, *et al.*, "Using simulation to explore distributed key-value stores for exascale systems services," in *Proceedings of the 2013 ACM/IEEE conference on Supercomputing*, SC '13, 2013.

[32] D. Zhao, *et al.*, "Towards high-performance and cost-effective distributed storage systems with information dispersal algorithms," in *IEEE International Conference on Cluster Computing*, IEEE CLUSTER '13, 2013.

[33] D. Zhao *et al.*, "HyCache: a user-level caching middleware for distributed file systems," International Workshop on High Performance Data Intsensive Computing, IEEE IPDPS '13, 2013.

[34] Intrepid https://www.alcf.anl.gov/resource-guides/intrepid-file-systems.

[35] Amazon Elastic Compute Cloud (Amazon EC2), " http://aws.amazon.com/ec2/."

[36] B. Fitzpatrick, "Distributed caching with memcached," *Linux J.*, vol. 2004, pp. 5–, Aug. 2004.

[37] G. DeCandia, *et al.*, "Dynamo: amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, 2007.

[38] Y. Gu *et al.*, "Supporting configurable congestion control in data transport services," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, 2005.

[39] D. Capps, "IOzone Filesystem Benchmark," *www.iozone.org*, 2008.

[40] Y. Zhao, *et al.*, "Swift: Fast, reliable, loosely coupled parallel computation," in *IEEE SCW*, pp. 199–206, 2007.

[41] M. Wilde, *et al.*, "Extreme-scale scripting: Opportunities for large task parallel applications on petascale computers," in *SCIDAC, Journal of Physics: Conference Series 180. DOI*, pp. 10–1088, 2009.

[42] Y. Zhao, *et al.*, "Opportunities and challenges in running scientific workflows on the cloud," in *Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, CYBERC '11, 2011.