Considerations for the Next Iteration of the DOE Fast Forward Storage and IO Project

Jay Lofstead
Sandia National Laboratories
gflofst@sandia.gov

Ivo Jimenez
University of California, Santa Cruz
ivo@cs.ucsc.edu

Carlos Maltzahn
University of California, Santa Cruz
carlosm@soe.ucsc.edu

ABSTRACT

The Fast Forward effort to define a next generation filesystem has made admirable contributions in architecture and design. Formalizing the general idea of data staging as burst buffers for the file system will help manage the performance variability and offer the data processing opportunities outside the main compute and file system. Adding a tranasctional mechanism to manage faults and data visilibilty helping to enable effective analytics without having to work around the IO stack semantics.

While these and other contributions are valuable, improvements can be incorporated. For example, the Doubly Distributed Transactions (D^2T) protocol offers an alternative approach for incorporating transactional semantics into the data path. The differing semantics between the two approaches and the functionality provided offer alternatives. Further, the lessons learned in implementing D^2T has generated insights into what the FastForward file system could and perhaps should do, given different features.

This paper examines some of the choices made by the FastForward team and compares them with other options and offers suggestions based on experiences.

Categories and Subject Descriptors

D.4 [**Software**]: Operating Systems; D.4.7 [**Operating Systems**]: Organization and Design—*hierarchical design*

General Terms

Design, Performance

1. INTRODUCTION

Current production HPC IO stack design is unlikely to offer sufficient features and performance to adequately serve the needs of an extreme scale platform. To address these limitations, the US Department of Energy commissioned an effort to develop a design and prototype for an IO stack suitable for the extreme scale environment. This is a joint effort led by the Intel Lustre team, Los Alamos National Laboratory, EMC, and the HDF Group. This team has developed a specification for a future IO stack to address the identified challenges.

The basic architecture incorporates five layers. The top layer is a high level IO library, such as the demonstration HDF-5 library. Below that is an IO forwarding layer that redirects IO calls from the compute nodes to the IO dispatching layer. This IO forwarding layer is analogous to the function of the IO nodes in a BlueGene machine. The IO dispatcher (IOD) has considerable functionality.

The IOD serves as the sole interface between an application and the persistent storage array. The core idea for IOD is to provide a way to manage the IO load that is separate from the compute nodes and the storage array. Communication intensive acitivies, such as two-phase, data sieving IO's data rearrangement phase, can be moved to the IOD layer offloading the communication load from the compute nodes. IOD has three main purposes. First, if the optional burst buffer is availble, it works as a fast cache absorbing write write operations for the slower trickle out to the central storage array. It can also be used to retrieve objects from the central storage array for more efficient read operations. Second, it offers the transaction mechanism for controlling data set visibility prior to it being correct and complete. Third, data processing operations can be placed in the IOD. This operations are inteded to offer data rearrangement, filtering, and similar operations prior to it reaching the central storage array.

This offloads the requirement for collective two-phase data sieving on the compute nodes to reorganize data. This technique has been proven effective at reducing the total time for writing data due to fewer participants involved in the communication patterns [13].

The bottom two layers are the Data Access Object Storage (DAOS) and Virtual Object Storage Device (VOSD). DAOS is the typical interface that represents the individual objects stored including the semantics related to "files". VSOD is the actual storage mechanism for the objects defined at the DAOS layer.

Incorporated into the design of these layers and the function of some of these layers themsleves, such as the IOD layer, are features to manage both burst IO performance and faults. In particular, the IOD layer's use of burst buffers attempts to absorb a massive data dump from the compute area that is then trickled to the persistent storage area only

PDSW13 November 18, 2013, Denver, CO, USA

ACM 978-1-4503-2505-9/13/11...$15.00.
http://dx.doi.org/10.1145/2538542.2538567

if desired. Otherwise, it can be stored locally on optional SSDs for use by analysis routines eliminating the need to use slower rotational media as the intermediary between a simulation and the analysis code.

A second feature is the incorporation of a transaction like mechanism into the IOD layer and a related epochs feature into the DAOS layer. This feature offers a mechanism to avoid making incomplete data visible to other users and offer a level of protection if the data can be stored on the optional SSDs.

These features aim to address both the bursty nature of HPC IO and to make the data storage stack resilient to faults. While these features can address these concerns, a deeper evaluation of the design suggests exploring alternatives may improve the performance and/or functionality desired.

The rest of the paper is organized as follows. Section 2 discusses some of the features of incoporating burst buffers as designed and suggests some considerations and alternatives for the next generation of this project. Section 3 discusses the transactions approach offered in the IOD layer and offers a comparison to the D^2T system. Section 4 discusses some of the optimizations offered by the current design and how and when these optimizations will work best. It also explores some of the dark corners that may yield performance issues with the current design. Finally, Section 5 disucsses the system overall with recommendations on what design elements should be reconsidered based on broader issues with current HPC data centers.

2. BURST BUFFERS

The idea of burst buffers were initially explored in the context of data staging [2, 1, 15, 17] and later incorporated as part of the IO stack [5, 4]. The proposed optional use of SSDs is problematic. First, with these burst buffers being optional, the semantics of how IOD works must change to use DAOS to store the data directly rather than storing them locally until explicitly persisted by the user. How this work work, the impact on metadata, and the ability to support functionality such as data reordering, such as chainging the fast dimension of an array, is undefined. How doing function shipping into an IOD without a burst buffer would work is also not explored. Further thought about how to have an IOD layer both with and without a burst buffer is required before they can be considered optional. As the design stands today, they are a required part of the IOD layer for proper functioning.

3. TRANSACTIONS AND EPOCHS

The transaction mechanism manifests in two forms. At the IOD layer, they are called transactions and are used to judge whether or not an output is complete or not. At the DAOS layer, they are called epochs and represent persisted transactions from the IOD layer. Each of these offers different functionality, but are connected as is explained below.

3.1 IOD Transactions

To understand how transactions are used in the IOD layer, some terminology and concepts must be explained first. At the coarsest grain level is a container. Each container serves to host a transaction and contains a collection of objects. Conceptually, containers corresponds to a file in a traditional file system. The objects in each container represent different data within a file. The three initially defined object types are key-value store, array shard, and blob. The easiest way to understand these types is to evaluate these from the perspective of an HDF-5 file. The key-value store represents a collection of attributes. The array shard represents part of a potentially multi-dimensional array. It is refered to as a shard because it is likely a small piece of a globally defined array. The blob represents a byte stream. The fundamental difference between an array shard and a blob is that the array shard has metadata identifying its portion within the global, logical space while the blob is simply a 1-dimensional array of bytes that is not shared across IOD nodes. Should an operation be deployed into IOD to manipulate an array within a container, it would operate on the array shards rather than on blobs. Given this context, the transactions come in two forms.

First is a single leader transaction where the IOD manages based on calls from a single client. The underlying assumption is that the client side will manage the transactional operations itself and the single client is capable of reporting to the IOD how to evolve the transaction state.

The second form is called multi-leader and has the IOD layer manage the transactions. In this case, when the transaction is created, a count of clients is provided to the IOD layer. As clients write to the container, the reference count is reduced. Once the count reaches 0, the transaction is automatically committed.

3.2 DAOS Epochs

The Epoch mechanism differs from transactions. Instead of focusing on when a particular output is complete, an epoch represents incremental persisted copies of a container. Each persisted copy increments the epoch. NEED TO CORRECT THIS INFORMATION AS IT IS WRONG.

3.3 Metadata Management

While a central design goal of IOD and DAOS is to eliminate the metadata server as a core component, it was not fully accomplished. The purpose behind the goal is to eliminate the serialized bottleneck caused by having a centralized metadata service. Much of the design of IOD with objects and continers does work to eliminate these bottlenecks. However, IOD does not go fully to a no metadata service model. Instead, the IOD layer describes a key-value metadata service it provides. This service maintains four different items:

- The list of IOD objects within the container
- The mapping from each IOD object to the corresponding DAOS object.
- The sharding and striping of the IOD object
- The maximum valid offset of the object

While a pure no metadata model would be ideal for performance, this current proposal has a few challenges. First, there are no list of containers. This is closer to the no metadata ideal, but given the rest of the functionality, it exports to the user level a requirement to track what containers are in the file system. Second, given the flattening that occurs between IOD and DAOS, the mapping from IOD objects to DAOS objects is not likely to be 1-to-1. This is likely a list

instead. Further, given the ability to re-stripe in the IOD layer, the list is not a list of objects completely included in the IOD object, but instead objects from which part of the data came from some listed DAOS object. Third, sharding IOD objects makes sense based on the compute process connection a shard has with the complete object. Striping of an IOD object makes sense in connection with the DAOS object(s) underneath, but striping within the IOD layer is not discussed. Further, using striping from clients into the IOD layer introduces a level of coordination that IOD seeks to avoid. Fourth, given the previous items, there is ambiguity with the maximum valid offset of the object. This could be for the shard or for the collection of shard objects that represent a distributed variable.

The D^{2}T project created a simplistic model for a metadata service that has different features that is described below and has been published about previously [**?**]. Some of the major differences include the following. First, D^{2}T's metadata service has a way to get a list of the ojbjects in the metadata service. While it does not address having a file equivalence like the container concept, it does introduce a way to figure out what is available. Stored along with this object description is a list of the global dimensions for array objects. One observtion made as part of creating this service is that this is insufficient for many engineering codes that use non-regular meshes. Second, for each object, a list of the equivalent to shards are kept with a link back to the master object and the offset and size for each dimension for that shard. This approach is not scalable because of the explosion in the process count. Using a striping approach like IOD does is superior. Third, there is no mapping to another layer with different object layout and counts.

Based on the lessons from the D^{2}T metadata service construction, having a completely separate metadata service is workable. Rather than making it a bottleneck in the IO path, it is another service that users must interact with if they need those services. Otherwise, it is 100manage everything by maintaining the metadata including the list of objects themselves. However, there are drawbacks to this 100

For the IOD proposal, the service needs some adjustments to be generally useful. First, there needs to be a way to query the list of available containers. Second, the maximum transaction ID for a container is mentioned as being maintained by this metadata service, but when and how this is done is never mentioned. Second, it should be extracted from IOD entirely and kept as a separate service along side. This will eliminate any contention caused by interaction with the IOD layer. This also offers a way to link with the DAOS layer in a machine-independent way. The need for this is described below. Third, the definition of an object needs to be made more clear. In some cases, it seems that an object is a globally distributed object is represented by a collection of shards. In other cases, it is a single shard.

3.4 Comparison to D^{2}T

The D^{2}T project [12] sought to develop an efficient approach for handling ACID-style transactions in an environment with parallel clients and multiple servers (doubly distributed). Rather than being aimed solely at data movement operations, D^{2}T seeks to address the general problem of managing any operation with multiple clients and servers. Consider the management of the analysis/visualization area, potentially similar to the IOD concept. The transaction protocol is used to help manage resizing of the resource allocation to the various analysis and visualization components. For the purposes of this discussion, D^{2}T could also be used to manage changing how IOD processes and/or nodes are used without exposing these changes to the client processes prematurely. This has been described and analyzed previously [7].

To address the gap between pure D^{2}T and IOD's transactions, example data storage and metadata services were created. These examples were not built for performance but to determine the minimum functionality requirement for a functional service. While these services were built without knowledge of how IOD and DAOS operate, we came to similar functionality. The major difference is that there are no dependencies between transactions that prevent visibility should an older version be incomplete. This additional, intentional requirement by IOD offers different functionality than D^{2}T's example services. In the case of D^{2}T, the functionality is more minimal, but also avoids some of the concerns outlined below.

The second iteration of the protocol [11] fixed scalability issues and demonstrated a scalable client-side coordiation model with excellent performance. The performance measured for a complex transaction with D^{2}T is illustrated in Figure 1. This performance is explored in detail in a previous paper [11]. The breakdown of the number of participants in each role is shown in Table 1. For comparison, consider the Number of Sub-Coordinators equivalent to an IOD process. The Processes Per Sub-Coordinator represents the number of clients that use a particular IOD. For these tests, we maintained a balanced distribution and always used at least two sub-coordinators to slow down the processing.

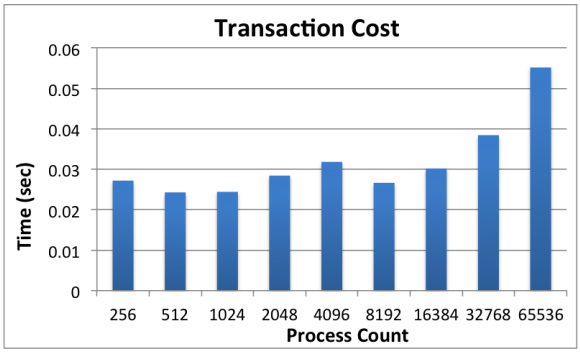

Figure 1: Total Transaction Overhead

Table 1: Performance Tests Scaling Configuration

Processes	Number of Sub-Coordinators	Processes Per Sub-Coordinator
256	2	128
512	2	256
1024	4	256
2048	8	256
4096	16	256
8192	32	256
16384	64	256
32768	128	256
65536	256	256

At a a high level, both D^{2}T and the IOD transactions have the same high-level design. In both cases, a hierarchical

model is employed. In the case of D^{2}T, it is a purely client-side tree. For IOD, it is a server-side tree. In both cases, there is a master in charge of managing the transaction and a collection of workers that aggregate into the master through second-level leaders. Beyond that, there are some significant differences. Some of the different choices made by IOD raise some possible concern.

First, D^{2}T has a timeout mechanism to detect failures and offer an ability to recover and clean up the incomplete transactions. IOD's transactions are asynchronous precluding a relatively simple, short timeout value to detect failures. Unfortunately, at this point, there is no mechanism in IOD's transaction handling to detect a fault and clean up any incomplete operations.

Second, should a transaction on a container not complete, all subsequent transactions on that container, even if they are complete and correct, will not be accessible.

Third, in the single leader model, if the process that manages the transaction were to fail, there is no ability to abort that transaction. The data will not be made available because the transaction is still listed as in progress. The problem is that without a way to clean up this failed transaction, no newer versions of the same container can be made readable. In reality, the single leader model would ideally outsource the transaction processing to something like D^{2}T on the client side.

Fourth, in the multi-leader model, using a count of client connections to determine if transaction is complete is problematic. First, should this be in an evnironment where lost or repeated network messages are possible, then the count could be wrong inappropriately. This is a general problem and unlikely in the current environment, but is a concern on less integrated platforms. The idea of a count precludes any tracking of expected messages from sources introducing a potential hole. Second, the aggregation of completion messages are based on the local aggregation point reporting to the transaction leader the state. There is no ability to detect or recover from a failure of this IOD process that would report to the transaction leader. Third, because a simple count of participating processes is sent with the beginning of the transaction, each process is strictly limited to a single output operation. This precludes a process writing multiple array shards, for example. To put this in concrete terms, it means that a file could only contain a single, globally distributed variable (at the root level of HDF5) OR an attribute per process. If this count actually represents a begin/end transaction pair instead, it suffers from a lack of accountability for missing object (shard) writes. All it would indicate is that a process started and ended a transaction connection.

D^{2}T has addressed this count issue in a couple of ways. First, the sub-coordinators each have a list of processes from which they expect messages. Should a message be missed, it is noticed and corrective action can be taken. Second, D^{2}T has the concept of sub-transactions. The messaging requirements are illustrated in Figure 2. Sub-transactions represent finer grained operations than the entire output, D^{2}T can manage multiple writes per client by using a sub-transaction to represent the output for any item to the file (container). Because of how the sub-transactions are managed, the singleton sub-transactions must be declared before the transaction begins ensuring there is global knowledge that the sub-transaction is expected. That way if the coordinator (transaction leader) fails, which ever process takes over that role knows to expect a completion message for that sub-transaction or the overall transaction cannot complete. While this additional layer does introduce messaging, the overhead is quite small.

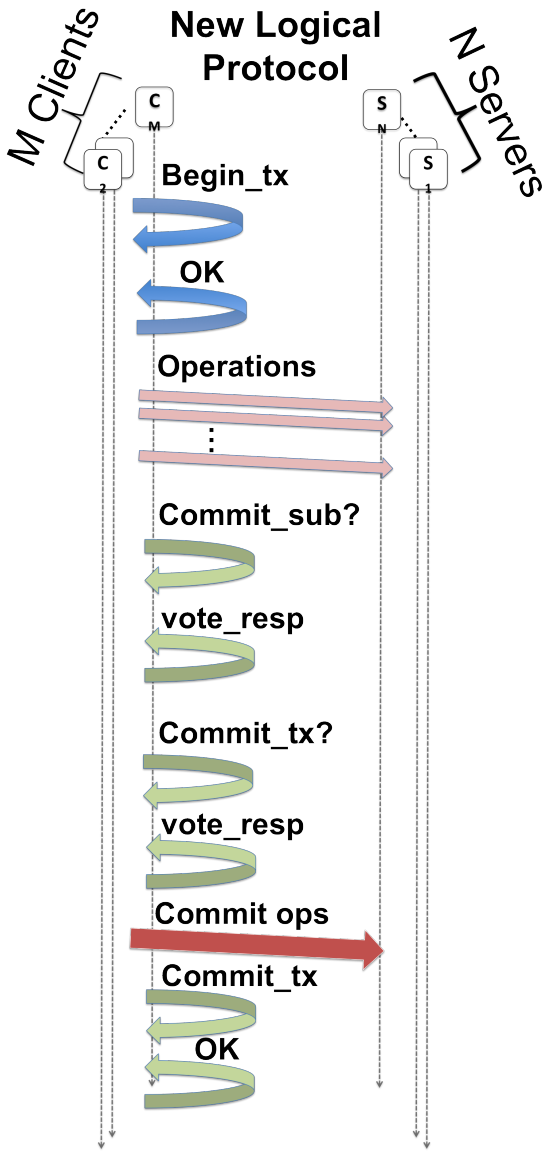

Figure 2: Optimized Protocol

3.5 Comparision to Other Protocols

Alternatives, such as Paxos [10] algorithms like ZooKeeper [9], suffer from two limitations making them unsuitable for this environment. First, the distributed servers in Paxos systems are all distributed copies of each other that eventually become consistent. Given the scale we wish to address, a single node's memory is unlikely to be able to hold all of the data necessary for many operations at scale. They also do not have a guaranteed for when consensus will be achieved without using slower synchronous calls. For the tight timing we wish to support, we need guarantees of when a consistent state has been achieved. Second, these systems also all assume that updates are initiated from a single client process rather than a parallel set of processes as is the standard

in HPC environments. The Intel/Lustre FastForward epoch approach [3, 14] is discussed in Section **??**.

Third, the server must offer a way to mark an operation as "committed" during the "commit" phase of the two-phase commit. Again, this may be implemented in a transactional wrapper. As an alternative, a scalable locking service like Chubby [6] might be employed.

Fourth, a second layer of coordination on the client side is introduced that greatly increases the scalability by consolidating messages from clients into unique sets prior to sending to the overall coordinator. A gossip protocol [8] may appear sufficient for this purpose, but the delay of eventual consistency is strictly avoided with this protocol to ensure guarantees at particular states in the code. For example, if a failure occurs, the global understanding of the role of all processes is required in order for effective communication to occur for operations like creating sub-transactions or voting. In this case, the protocol can offer stronger statements about consistency than these protocols offer. These features offer a way to easily scale the transaction protocol given the guarantees we wish to offer.

Another effort to offer consistency and data integrity for the ZFS file system [16] covers some of the same territory. Instead of a focus on the processes all having a notion of completion as a transaction, this work focuses on the integrity of the data movement operations. We view this work as something that should be considered hand-in-hand with a transaction approach to ensure the integrity of the movement of the data in addition to the agreement of processes about the successful completion of a parallel operation.

4. OPTIMIZATIONS

The ability to place functionality at the IOD layer was explored earlier. In particular, PreDatA [17] demonstrated the advantage to the approach as well as identified that the placement of the operators should consider the compute nodes, the staging or burst buffer nodes, or offline depending on the characteristics of the operation. This consideration should be incorporated into the design. In particular, one of the key observations is that the reduced process count may require the entire time between output operations to perform many of the data processing tasks requested. This was part of the motivation prompting evaluating placement decisions at the compute node, in a staging area, or offline. In the case of the IOD design, it forces a staging area deployment and then shares that staging area across all users simultaneously. This is unlikely to be useful because of the limited compute and communication capacity to spare to perform these operations at a bottleneck in the IO path. The use of a separate staging area to perform these computations intentionally selects a location separate from the IO path to avoid these bottlenecks and potential scalability issues.

5. BROADER DESIGN

Consider a shared file system across an HPC data center. The current design maintains the metadata in the IOD layer localized to a single machine effectively making the data inaccessible from another platform.

There is a change in definitions between the IOD layer and the DAOS layer. For the IOD layer, a container is a collection of objects. For the DAOS layer, a continer is a collection of shards. For the IOD an object may be a shard of a global array. For DAOS, a shard can host a set of DAOS objects. Having the same names with locally correct, globally conflicting definitions serves to confuse how the system should work.

6. CONCLUSIONS

We have demonstrated that synchronous two-phase commit transactions can have low overhead and be used with roughly synchronous operations with little additional overhead. While other use cases, particularly those that require a greater degree of asynchrony, require a different approach, D^2T offers a working solution for a wide variety of scenarios today. The bigger question of what sort of synchronization mechanism is appropriate for various use cases is currently under investigation.

The evaluation of the fault detection and recovery are beyond scope for this paper and are in preparation for presentation elsewhere.

7. ACKNOWLEDGEMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

8. REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan, and M. Wolf. Extending i/o through high performance data services. In *Cluster Computing*, Luoisiana, LA, September 2009. IEEE International.

[2] H. Abbasi, M. Wolf, and K. Schwan. LIVE data workspace: A flexible, dynamic and extensible platform for petascale applications. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 341–348, Washington, DC, USA, 2007. IEEE Computer Society.

[3] E. Barton. Lustre* - fast forward to exascale. Lustre User Group Summit 2013, March 2013.

[4] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[5] J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, and A. Torrez. Storage challenges at los alamos national lab. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[6] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In B. N. Bershad and J. C. Mogul, editors, *OSDI*, pages 335–350. USENIX Association, 2006.

[7] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. Lofstead. I/o containers: Managing the data analytics and visualization pipelines of high end codes.

In *In Proceedings of International Workshop on High Performance Data Intensive Computing (HPDIC 2013) held in conjunction with IPDPS 2013*, Boston, MA, 2013. Best Paper Award.

[8] A. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *Computers, IEEE Transactions on*, 52(2):139–149, 2003.

[9] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *In USENIX Annual Technical Conference*, 2010.

[10] L. Lamport and K. Marzullo. The part-time parliament. *ACM Transactions on Computer Systems*, 16:133–169, 1998.

[11] J. Lofstead, J. Dayal, I. Jimenez, and C. Maltzahn. Efficient transactions for parallel data movement. In *The Petascale Data Storage Workshop at Supercomputing*, Denver, CO, November 2013.

[12] J. Lofstead, J. Dayal, K. Schwan, and R. Oldfield. D2t: Doubly distributed transactions for high performance and distributed computing. In *IEEE Cluster Conference*, Beijing, China, September 2012.

[13] J. Lofstead, R. Oldfiend, T. Kordenbrock, and C. Reiss. Extending scalability of collective io through nessie and staging. In *The Petascale Data Storage Workshop at Supercomputing*, Seattle, WA, November 2011.

[14] J. Lombardi. High level design - epoch recovery, june 25th, 2013. Intel FastForward Wiki, June 2013.

[15] A. Nisar, W.-k. Liao, and A. Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[16] Y. Zhang, A. Rajimwale, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. End-to-end data integrity for file systems: A zfs case study. In R. C. Burns and K. Keeton, editors, *FAST*, pages 29–42. USENIX, 2010.

[17] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on Peta-Scale machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.