Building a parallel file system simulator

E Molina-Estolano[1], C Maltzahn[1], J Bent[2] and S A Brandt[1]

[1] UCSC Systems Research Lab, University of California, Santa Cruz, CA 95064, USA
[2] Los Alamos National Lab, Los Alamos, NM 87545, USA

E-mail: {carlosm,eestolan,scott}@cs.ucsc.edu, johnbent@lanl.gov

Abstract. Parallel file systems are gaining in popularity in high-end computing centers as well as commercial data centers. High-end computing systems are expected to scale exponentially and to pose new challenges to their storage scalability in terms of cost and power. To address these challenges scientists and file system designers will need a thorough understanding of the design space of parallel file systems. Yet there exist few systematic studies of parallel file system behavior at petabyte- and exabyte scale. An important reason is the significant cost of getting access to large-scale hardware to test parallel file systems. To contribute to this understanding we are building a parallel file system simulator that can simulate parallel file systems at very large scale. Our goal is to simulate petabyte-scale parallel file systems on a small cluster or even a single machine in reasonable time and fidelity. With this simulator, file system experts will be able to tune existing file systems for specific workloads, scientists and file system deployment engineers will be able to better communicate workload requirements, file system designers and researchers will be able to try out design alternatives and innovations at scale, and instructors will be able to study very large-scale parallel file system behavior in the class room. In this paper we describe our approach and provide preliminary results that are encouraging both in terms of fidelity and simulation scalability.

1. Introduction

We develop a novel approach to building a simulator for parallel file systems. Parallel file systems using object-based storage devices have been the storage architecture of choice as the scale of scientific high-end computing systems increases exponentially [1]. There are a handful of parallel file systems in use today, namely Lustre [2], PanFS [3], and PVFS [4], that are all based on the object-based storage paradigm [5] popularized by NASD [6]. New variations of parallel file system architectures have been proposed or are in development: Ceph [7], zFS [8], Sorrento, Ursa Minor [9], Kybos [10]. There is tremendous pressure to improve the performance of these systems. Yet, presently the performance of these systems cannot be easily tested without access to large clusters and expensive compute time. Existing large-scale systems are generally reserved for production use, running scientific codes 24/7, with very limited opportunities for other uses. Analytical models and simulators that are carefully validated by the high-end performance community would allow testing of file system designs at scale using a tiny fraction of the resources required by real system tests.

In this paper we introduce IMPIOUS (Imprecisely Modeling Parallel I/O is Usually Successful), a trace-driven parallel file system simulator using an *abstract simulation model* of object-based parallel file systems that accounts for just enough detail to reproduce important effects observed in real systems. IMPIOUS will (1) enable file system designers and researchers to try out innovative data placement strategies and other novel subsystems at scale, (2) facilitate file system deployment by providing a low-cost platform for "what-if" workload and file system tuning scenarios, (3) empower scientists to quickly

tune existing file systems for specific workloads, and (4) aid instructors by providing a platform for in-classroom experiments.

The benefit of abstract simulation models is their simplicity, which allows users to simulate very large systems on much smaller systems within an acceptable amount of time. Our central hypothesis is that an abstract model with a few simple components suffices to describe important aggregate behaviors of parallel filesystem without having to model detailed behavior. For example, a simple model of disks works as well as detailed models as used in DiskSim [11].

The only parallel file system simulator we are aware of is the one reported in [12]. That simulator uses a detailed model of PVFS and Linux using the OMNeT++ simulation framework [13], focuses on network communications among OSDs by using an accurate simulation of TCP/IP over Ethernet, and is tuned to the particular infrastructure of the author's institution. Thus, the authors have chosen a what we call "bottom-up" strategy in that they first strive to model the behavior of the underlying systems with high fidelity before combining these simulation models to simulate a parallel file system. The disadvantage of this bottom-up approach is that it introduces a large amount of detail into the simulation which lengthens simulation run time, makes it more difficult to separate behavior due to a specific file system environment from behavior due to fundamental file system design decisions, and requires greater effort to change and experiment with alternative simulation models and therefore discourages rapid hypothesis testing.

In this project we hope to achieve the following contributions: (1) *a simple, scalable, and general simulation model* – the model supports file system-specific plugins and consists of about 10 general model parameters such as number of clients, number of OSDs, page size, and network latency; (2) *a framework for rapid hypothesis testing* – the simulation model can be refined easily to include a hypothesis for a particular behavior of a parallel file system for a particular workload; (3) *a simulator* – we are planning to make IMPIOUS available to the open source community. It currently consists of about 10k lines of Python code.

2. An Abstract Model of Parallel File Systems

Parallel file systems are complex systems, and the details of the implementations of commonly used systems differ significantly. Yet, they consist of the same subsystems: *object-based storage devices (OSDs)* which run their own local file system and export a flat name space of objects; *metadata management* which implements the file name space and all metadata operations; and *clients*, which provide the (typically POSIX) file system interface to applications. Clients communicate with metadata management to stat/open/close files and manipulate the name space, and directly access OSDs to read and write data.

The key characteristics distinguishing particular parallel file systems are the *data placement strategy*, which maps a request to the client to requests to one or more OSDs using a particular striping strategy and a redundancy scheme; the *resource locking protocol*, which determines which disks or OSDs are locked when for a particular client (e.g. locking the entire RAID group when accessing a stripe in that group); the *redundancy strategy*, which determines the kind of redundancy and which subsystem is responsible for maintaining it (e.g. clients, OSDs, or disk controllers); and the *client buffer cache*, which specifies management of communication between clients and OSDs (e.g. asynchronous vs. synchronous and arbitrary extents vs. pages).

3. The IMPIOUS Simulator

The simulator uses a plugin for the desired parallel filesystem, implementing the above components. In a simulation run, a set of simulated clients is driven by a parallel file I/O trace or by a workload generator. (We currently have generators for N-N and N-1 checkpointing workloads.) The clients produce I/O requests to send to the OSDs, following the plugin behavior. Each simulated OSD consists of a simple internal filesystem, which we call NaiveFS; and a simulated disk, with the option of using either DiskSim or a simple disk model.

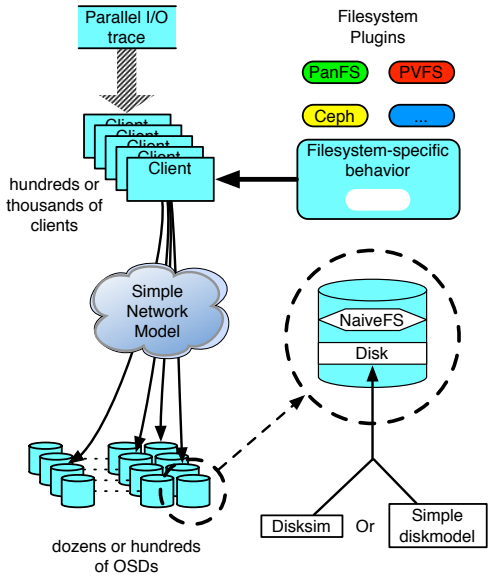


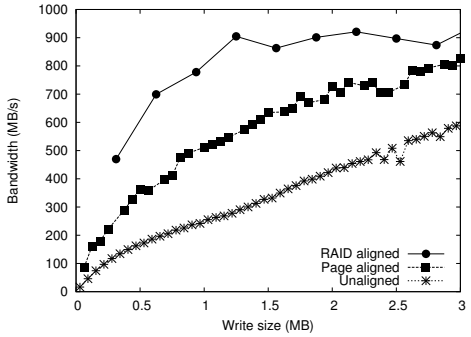
Figure 1: **The major components of IMPIOUS.** In the simulator, clients are driven by parallel I/O traces. Filesystem-specific plugins determine certain fundamental behavior: the data placement strategy, replication strategy, locking discipline, and client cache strategy. Clients communicate with OSDs to perform I/O. Each simulated OSD consists of an instance of NaiveFS, an idealized per-OSD filesystem; and a disk simulation, which can be either DiskSim or a simple disk model.

We have implemented the models for PVFS, PanFS, and Ceph; and we are in the process of implementing Lustre. We chose these file systems because they are commonly used (except Ceph), we have access to their placement strategies, and their placement strategies are sufficiently different. Our preliminary validation results are based on PanFS [3].

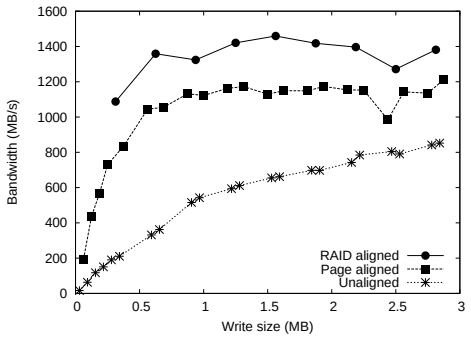
PanFS is the filesystem used in the Panasas parallel storage product. Its data placement uses per-file RAID 5 groups, with a fixed per-filesystem stripe width of 8 to 11 OSDs. For each file, a randomized algorithm (biased towards emptier nodes) selects the OSDs for the first RAID-5 stripe set; data is written repeatedly across the stripe set, in chunks of 64KB per OSD for each stripe. Up to 2,000 such stripes are written across the stripe set; once that limit has been reached, another stripe set, disjoint from the first one, is selected. If the file grows sufficiently that there are not enough OSDs to create another disjoint stripe set, and all stripe sets have been written across 2000 times, writes continue through the existing write sets in a round-robin fashion, again with 2000 writes per stripe set. To keep the parity consistent, a client must acquire an exclusive lock *per stripe* in order to write to it. This degrades performance for multiple-writer workloads with small interleaved writes. To handle node failure, PanFS uses *virtual spares* in conjunction with RAID-5. The stripe length for the filesystem is selected so that the maximum number of stripe sets per file will leave a small number of spare OSDs. Thus, if a node fails, each file with a stripe set on that node can replace that position in the stripe set with a spare node, and reconstruct the data from parity.

4. Preliminary Validation Results

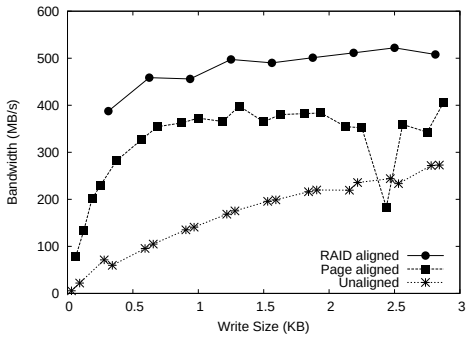
In order to validate IMPIOUS, we use traces of important simulation checkpointing patterns [14]. The first experiment attempts to reproduce the effect of alignment on write bandwidth for N-1 checkpoint patterns; we use experimental N-1 checkpointing results collected using the PatternIO benchmark [15] running on 512 processes on LANL's Roadrunner test cluster, which includes a PanFS storage system consisting of 56 OSD's. These results are shown in Figure 2a: the three lines in this graph illustrate the



(a) Experimental Results



(b) IMPIOUS Simulated Results (DiskSim)



(c) IMPIOUS Simulated Results (simple disk model)

Figure 2: **N-1 Write Alignment** Graph (a) shows how checkpoint write bandwidth for an N-1 workload is affected by the alignment of writes. The three lines show writes that are aligned with stripe lengths, writes that are not aligned with stripe length but are page-aligned, and unaligned writes. We run IMPIOUS with both DiskSim (b) and the simple disk model (c). The graphs show that IMPIOUS is capable of approximating the performance differences among these different sets of write sizes (we are still investigating the pronounced dip at the 2496 KB chunk size under the simple disk model).

effect of write size on write bandwidth. All three lines improve as the writes get larger.

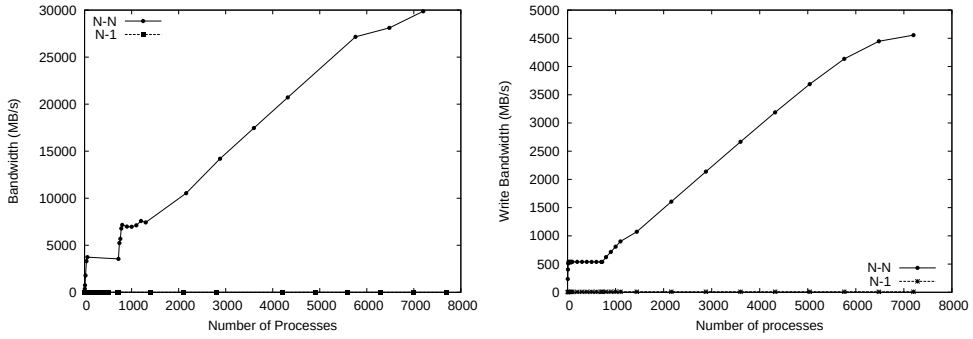
The top line shows the bandwidth achieved when the size of the writes is a multiple of the full stripe length (10 OSDs or 640K in this case); this achieves the highest bandwidth. The middle line shows the bandwidth for page aligned write sizes (4K in this case) which are not a multiple of the full stripe length. Writing partial stripes is slower for two reasons. First, each partial write is a read-update-write cycle, since existing data from the stripe is needed to compute the parity. Second, since the stripe is locked during each write, any other writers trying to write to the same stripe are blocked. Finally, the bottom line shows the bandwidth for write sizes that are not a multiple of the 4K page size.

These unaligned writes achieve the least bandwidth for two reasons. First, the PanFS client synchronously flushes dirty data that is smaller than a page. For example, compare a 32K write to a 31K write, both beginning at the start of a page. PanFS will buffer the 32K write until memory pressure forces it to be flushed, at which point it will write the entire 32K at once. Conversely, it will split the 31K write into 28K and 3K writes. Thus, the 31K write will incur an extra round trip to the OSDs, and an extra parity update. Second, an idiosyncrasy of the experimental run improves the performance of the aligned writes, but not the unaligned writes. The 512 processes were placed in groups of eight consecutive processes per compute node, with each group sharing a single PanFS client and buffer cache. This allowed each group's writes to be aggregated; for instance, the workload of 512 processes performing 128K writes was effectively a workload of 64 clients performing 1024K writes. This aggregation could

not happen for unaligned write sizes, because of the synchronous flushing behavior described above.

Figure 2b illustrates how IMPIOUS accurately models these trends. It does not match the absolute throughput values: we used DiskSim with an available recent disk model to model the OSDs' disks, and made no attempt to match the OSDs' precise performance characteristics. IMPIOUS correctly reproduces the relative results, capturing the performance differences among the different types of write sizes.

Even though IMPIOUS does not reproduce the absolute throughput values, the relative results are still very useful for a user wondering what sort of change they might expect if they were able to manipulate either their application or the underlying storage configuration to match the size of their application's write to the full stripe size of their storage system. Except for DiskSim these results are obtained with very simple simulation models. In fact, the bulk of the simulation's runtime is spent in DiskSim. To make IMPIOUS more scalable, we are developing a simple disk model, intended to give accurate *aggregate* results without precisely modeling the disk's internals. Figure 2c shows results with IMPIOUS using the simple disk model instead of DiskSim. With the exception of the pronounced dip with a chunk size of 2496K, which we are still investigating, using the simple disk model in IMPIOUS instead of DiskSim produces similar qualitative results, showing the three trend lines, while reducing the simulation's runtime by an order of magnitude. However, the absolute throughputs are considerably lower. We are still refining the simple disk model; we hope ultimately to use DiskSim parameters in the simple disk model, and match the aggregate behavior of DiskSim disks without the large runtime.



(a) Experimental Results (b) Simulated Results

Figure 3: **Checkpoint Types.** These two graphs depict the challenge to a parallel file system caused by an N-1 checkpoint pattern. Checkpoint write bandwidth is shown on the y-axis as a function of the number of processes. Due to locking, N-1 checkpoints have been demonstrated across multiple HPC parallel file systems to perform much more poorly than N-N. The graph on the left contains experimental data confirming this from the PanFS storage system attached to LANL's Roadrunner supercomputer. IMPIOUS is able to correctly predict the relative ordering of these two lines, and the linear growth of the N-1 throughput.

We also simulate the detrimental performance effects of an N-1 checkpoint pattern. The locking present in an N-1 checkpoint pattern, from multiple clients writing to the same file, significantly degrades performance compared to an otherwise similar N-N workload. We simulated a checkpointing experiment that was run on LANL's Roadrunner supercomputer, using a PanFS storage system of 512 OSDs. In this workload, a number of clients generate an N-1 strided workload in 47 KB writes.

Figure 3b shows the experimental Roadrunner results and our simulated results. IMPIOUS correctly captures the performance degradation caused by the N-1 workload. Each data point was produced by a simulation of 512 OSDs running on an 2 GHz Opteron, using only one core. With the aforementioned simple disk model, the simulation runtime varied from 12 minutes to 3 hours, scaling approximately

linearly with the total number of I/O operations. Thus, the entire set of simulations producing this graph can be run in under a day on a single machine, or in a few hours on a modestly sized compute cluster by running simulations concurrently.

However, the simulated absolute throughput values for the N-N workload are much lower than the Roadrunner results, and the bandwidth saturates at a low number of processes. According to our current simulation model implementation each 47KB write is treated separately, which increases communication overhead and incurs a large number of extra seeks. We hypothesize that PanFS contains an optimization where a client that is the only writer to a file coalesces multiple unaligned requests before writing them. We are in the process of implementing client level request coalescing in IMPIOUS ; as an approximation, we ran an N-N workload with the stripe-aligned write size of 576KB, which increased the throughput considerably.

The linear curve of the experimental N-N performance is due to an idiosyncrasy of the real system's architecture: The Roadrunner cluster is partitioned into sub-clusters (*CU*s), each supporting up to 720 processes, and each with an equal fraction of the total I/O bandwidth. This leads to a stair step function where each time the increasing number of processes on the x-axis spills over to a new CU, the total I/O bandwidth increases by roughly the fraction of the I/O bandwidth available to the new CU (Figure 3a appears to show a linear curve but at sufficient resolution the entire curve would reveal its stair step shape). We implemented the I/O bandwidth partitioning in IMPIOUS to capture this effect; without it, the throughput would rise asymptotically.

5. Conclusions and Future Work

Our preliminary simulation results are encouraging. Given two different sets of experimental results from checkpointing workloads on PanFS, we were able to simulate the important performance effects of those workloads with IMPIOUS . The wall-clock runtime IMPIOUS is also encouraging. Simulation runs required far fewer computing resources than running the original workload: for experimental runs consuming 2 minutes across a 512-node storage cluster, the slowest simulated runs took three hours on a single core.

We will continue to improve and validate IMPIOUS . To do so, we will acquire experimental results from different workloads. We wish to demonstrate IMPIOUS reproducing results from considerably different workloads on the same filesystem; and show it reproducing the performance differences of a single workload running on different filesystems.

References

[1] Top500 2008 Performance development http://top500.org/ URL `http://top500.org/lists/2008/11/performance_development`

[2] Schwan P 2003 *Proceedings of the 2003 Linux Symposium*

[3] Welch B, Unangst M, Abbasi Z, Gibson G, Mueller B, Small J, Zelenka J and Zhou B 2008 *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)* pp 17–33

[4] Carns P H, Ligon W B, Ross R B and Thakur R 2000 *Proceedings of the 4th Annual Linux Showcase and Conference* (Atlanta, GA) pp 317–327

[5] Azagury A, Dreizin V, Factor M, Henis E, Naor D, Rinetzky N, Rodeh O, Satran J, Tavory A and Yerushalmi L 2003 *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies* pp 165–176

[6] Gobioff H, Gibson G and Tygar D 1997 Security for network attached storage devices Tech. Rep. TR CMU-CS-97-185 Carniege Mellon

[7] Weil S A, Brandt S A, Miller E L, Long D D E and Maltzahn C 2006 *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)* (Seattle, WA)

[8] Rodeh O and Teperman A 2003 *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies* pp 207–218

[9] Abd-El-Malek M, II W V C, Cranor C, Ganger G R, Hendricks J, Klosterman A J, Mesnier M, Prasad M, Salmon B, Sambasivan R R, Sinnamohideen S, Strunk J D, Thereska E, Wachs M and Wylie J J 2005 *Proceedings of the 3nd USENIX Conference on File and Storage Technologies (FAST'05)* (San Francisco, CA)

[10] Wong T M, Golding R A, Glider J S, beth Borowsky E, Becker-Szendy R A, Fleiner C, mana Hosekote D R K and

Zaki O A 2005 Kybos: self-management for distributed brick-based storage Research Report RJ 10356 IBM Almaden Research Center

[11] Bucy J S, Schindler J, Schlosser S W, Ganger G R and Contributors 2008 The disksim simulation environment version 4.0 reference manual Tech. Rep. CMU-PDL-08-101 Parallel Data Laboratory, Carnegie Mellon University Pittsburgh, PA

[12] Carns P H, Settlemyer B W and Ligon W B 2008 *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (Piscataway, NJ, USA: IEEE Press) pp 1–8 ISBN 978-1-4244-2835-9

[13] Varga A 2001 *Proceedings of the European Simulation Multiconference*

[14] Bent J 2009 PLFS Maps http://www.institutes.lanl.gov/plfs/maps

[15] National Energy Research Scientific Computing Center I/O Patterns from NERSC Applications https://outreach.scidac.gov/hdf/NERSC_User_IOcases.pdf