Investigation Of Leading HPC I/O Performance Using A Scientific-Application Derived Benchmark

Julian Borrill, Leonid Oliker, John Shalf, Hongzhang Shan
CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720
{jdborrill,loliker,jshalf,hshan}@lbl.gov

ABSTRACT

With the exponential growth of high-fidelity sensor and simulated data, the scientific community is increasingly reliant on ultrascale HPC resources to handle their data analysis requirements. However, to utilize such extreme computing power effectively, the I/O components must be designed in a balanced fashion, as any architectural bottleneck will quickly render the platform intolerably inefficient. To understand I/O performance of data-intensive applications in realistic computational settings, we develop a lightweight, portable benchmark called MADbench2, which is derived directly from a large-scale Cosmic Microwave Background (CMB) data analysis package. Our study represents one of the most comprehensive I/O analyses of modern parallel filesystems, examining a broad range of system architectures and configurations, including Lustre on the Cray XT3 and Intel Itanium2 cluster; GPFS on IBM Power5 and AMD Opteron platforms; two BlueGene/L installations utilizing GPFS and PVFS2 filesystems; and CXFS on the SGI Altix3700. We present extensive synchronous I/O performance data comparing a number of key parameters including concurrency, POSIX- versus MPI-IO, and unique- versus shared-file accesses, using both the default environment as well as highly-tuned I/O parameters. Finally, we explore the potential of asynchronous I/O and quantify the volume of computation required to hide a given volume of I/O. Overall our study quantifies the vast differences in performance and functionality of parallel filesystems across state-of-the-art platforms, while providing system designers and computational scientists a lightweight tool for conducting further analyses.

1. INTRODUCTION

As the field of scientific computing matures, the demands for computational resources are growing at a rapid rate. It is estimated that by the end of this decade, numerous mission-critical applications will have computational requirements that are at least two orders of magnitude larger than current levels [10, 20]. To address these ambitious goals, the high-performance computing (HPC) community is racing towards making petascale computing a reality. However, to utilize such extreme computing power effectively, all system components must be designed in a balanced fashion to effectively match the resource requirements of the underlying application, as any architectural bottleneck will quickly render the platform intolerably inefficient. In this work, we address the I/O component of the system architecture — a feature that is becoming progressively more important for many scientific domains that analyze the exponentially growing volume of high-fidelity sensor and simulated data.

In this paper, we examine I/O performance behavior across several leading supercomputing systems and a comprehensive set of parallel filesystems, using a lightweight, portable benchmark called MADbench2. Unlike most of the I/O microbenchmarks currently available [11–13, 15, 17], MADbench2 is a unique in that it is derived directly from an important scientific application, specifically in the field of Cosmic Microwave Background data analysis. Using an application-derived approach allows us to study the architectural system performance under realistic I/O demands and communication patterns. Additionally, any optimization insight gained from these studies, can be fed back both directly into the original application, and indirectly into applications with similar I/O requirements. Finally, the tunable nature of the benchmark allows us to explore I/O behavior across the parameter space defined by future architectures, data sets and applications.

The remainder of the this paper is organized as follows. Section 2 motivates and describes the details of the MADbench2 code. Next, Section 3 outlines the experimental testbed, as well as the extensive set of studied HPC platforms and underlying parallel filesystems, including: Lustre on the Cray XT3 and Intel Itanium2 cluster; GPFS on IBM Power5 and AMD Opteron platforms; two BlueGene/L installations utilizing GPFS and PVFS2 filesystems; and CXFS on the SGI Altix3700. We then present detailed synchronous I/O performance data in Section 4, comparing a number of key parameters, including concurrency, POSIX-versus MPI-IO, and unique versus shared file accesses. We also evaluate optimized I/O performance by examining a variety of filesystem-specific I/O optimizations. The potential of asynchronous behavior is explored in Section 5, where we examine the volume of computation that could be hidden behind effective I/O asyncronicity using a novel system metric. Finally, Section 6 presents the summary of our findings.

Overall our study quantifies the vast differences in perfor-

SC07 November 10-16, 2007, Reno, Nevada, USA

mance and functionality of parallel filesystems across state-of-the-art platforms, while providing system designers and computational scientists a lightweight tool for conducting further analyses.

2. MADBENCH2

MADbench2 is the second generation of a HPC benchmarking tool [3, 5] that is derived from the analysis of massive CMB datasets. The CMB is the earliest possible image of the Universe, as it was only 400,000 years after the Big Bang. Measuring the CMB has been one of the Holy Grails of cosmology — with Nobel prizes in physics being awarded both for the first detection of the CMB (1978: Penzias & Wilson), and for the first detection of CMB fluctuations (2006: Mather & Smoot). Extremely tiny variations in the CMB temperature and polarization encode a wealth of information about the nature of the Universe, and a major effort continues to determine precisely their statistical properties. The challenge here is twofold: first the anisotropies are extraordinarily faint, at the milli- and micro-K level on a 3K background; and second we want to measure their power on all angular scales, from the all-sky to the arcminute. To obtain sufficient signal-to-noise at high enough resolution we need to gather — and then analyze — extremely large datasets. High performance computing has therefore become a cornerstone of CMB research, both to make pixelized maps of the microwave sky from experimental time-ordered data, and to derive the angular power spectra of the CMB from these maps.

2.1 Computational Structure

The Microwave Anisotropy Dataset Computational Analysis Package (MADCAP) has been developed specifically to tackle the CMB data analysis challenge on the largest HPC systems [2]. The most computationally challenging element of this package, used to derive spectra from sky maps, has also been re-cast as a benchmarking tool; all the redundant scientific detail has been removed, but the full computational complexity — in calculation, communication, and I/O – has been retained. In this form, MADbench2 boils down to four steps:

- Recursively build a sequence of Legendre polynomial based CMB signal pixel-pixel correlation component matrices, writing each to disk (loop over {calculate, write});
- Form and invert the full CMB signal+noise correlation matrix (calculate/communicate);
- In turn, read each CMB signal correlation component matrix from disk, multiply it by the inverse CMB data correlation matrix (using *PDGEMM*), and write the resulting matrix to disk (loop over {read, calculate/communicate, write});
- In turn, read each pair of these result matrices from disk and calculate the trace of their product (loop over {read, calculate/communicate}).

Due to the large memory requirements of the computational domain in real calculations, all the required matrices generally do not fit in memory simultaneously. Thus, an out-of-core algorithm is used, which requires enough memory to store just five matrices at any one time, with the individual component matrices being written to disk when they are first calculated, and re-read whenever they are required. Since the heart of the calculation is dense linear algebra, all the matrices are stored in the ScaLAPACK [19] block-cyclic distribution, and each processor simultaneously writes/reads its share of each matrix in a single binary dump/suck.

In order to keep the data dense (and therefore minimize the communication overhead) even on large numbers of processors, the implementation offers the option of gang-parallelism. In gang-parallelism the first two steps are performed on matrices distributed across all processors; however, for the last two steps, the matrices are remapped to disjoint subsets of processors (gangs) and the independent matrix multiplications or trace calculations are performed simultaneously. An analysis of this approach at varying concurrencies and architectural platforms, which highlights the complex interplay between the communication and calculation capabilities of HPC systems, is discussed in previous studies [3, 5].

2.2 Parameterized Environment

In this work, we present MADbench2, an updated version of the benchmark that has been extended in a number of ways to broaden its ability to investigate I/O performance both within and across HPC systems. MADbench2 can be compiled and run in I/O-mode — the mode used throughout here — in which all computations are replaced with busy-work whose scaling with data size can be tuned by the user. This tuning takes the form of an exponent α such that, for N bytes of data read or written, the code performs N^{α} floating-point operations. where all computations are replaced with busy-work. This allows MADbench2 to investigate I/O performance on experimental systems where the usual linear algebra libraries are not yet installed, and to run very large I/O performance tests without having to wait for redundant linear-algebra calculations. In addition, with asynchronous I/O, the busy-work exponent can be varied to measure how the I/O time changes with the scaling of the calculation behind which it is being backgrounded. In either mode, MADbench2 measures the time spent in calculation/communication and I/O by each processor for each step, and reports their average and extrema; in I/O mode the time spent in the busy-work function is measured and reported separately. Note that as yet the busy-work function includes no inter-processor communication.

MADbench2 accepts environment variables that define the overall I/O approach:

- IOMETHOD - either POSIX or MPI-IO data transfers,
- IOMODE - either synchronous or asynchronous I/O,
- FILETYPE - either unique or shared files (i.e. one file per processor versus one file for all processors), and
- BWEXP - the busy-work exponent α.

and command-line arguments that set the scale, gang-parallelism, and system-specific tuning:

- the number of pixels `NPIX`,
- the number of bins `NBIN`,
- the number of gangs, set to 1 throughout here,
- the ScaLAPACK block size, irrelevant here since we run in I/O-mode,
- the file blocksize `FBLOCKSIZE`, with all I/O calls starting an integer number of these blocks into a file,
- the read/write-modulators `RMOD/WMOD`, limiting the number of processes that read/write simultaneously.

of any particular analysis run. The first two arguments set the problem size, resulting in a calculation with NBIN component matrices, each of NPIX^2 doubles. The last three arguments are system-specific and are set by experiment to optimize each system's performance. This means that in code steps 1, 3 & 4 each processor will issue a sequence of NBIN read and/or write calls, each of O(8 NPIX^2/#CPU) bytes, and each fixed to start an exact multiple of FBLOCKSIZE bytes into the file. At the top level each of these I/O calls is issued by every processor at the same time, although only 1 in RMOD/WMOD low level read/write calls are actually executed simultaneously[1].

Note that although MADbench2 is originally derived from a specific cosmology application, its three patterns of I/O and work — looping over work/write (step 1), read/work/write (step 3), and read/work (step 4) — are completely generic. Since the performance of each pattern is monitored independently the results obtained here will have applicability to a very wide range of I/O-intensive applications. This is particularly true for data analysis applications, which cannot simply reduce their I/O load in the way that many simulation codes can by, for example, checkpointing less frequently or saving fewer intermediate states. MADbench2 also restricts its attention to I/O strategies that can be expected to be effective on the very largest systems, so each processor performs its own I/O (possibly staggered, if this improves performance) rather than having a subset of the processors read and scatter, or gather and write, data.

MADbench2 enables comparison between concurrent writes to shared files to the default approach of using one file per processor or per node. Using fewer files will greatly simplify the data analysis and archival storage. Ultimately, it would be conceptually easier to use the same logical data organization within the data file, regardless of how many processors were involved in writing the file. The majority of users continue to embrace the approach that each process uses its own file to store the results of its local computations. An immediate disadvantage of this approach is that after program failure or interruption, a restart must use the same number of processes. Another problem with one-file-per-processor is that continued scaling can ultimately be limited by metadata server performance, which is notoriously slow. A more serious problem is that this approach does not scale, and leads to a data management nightmare. Tens or hundreds of thousands of files will be generated on petascale platforms. A practical example [1] is that a recent run on BG/L using 32K nodes for a FLASH code generated over 74 million files. Managing and maintaining these files itself will become a grand challenging problem regardless of the performance. Using a single or fewer shared files to reduce the total number of files is preferred on large-scale parallel systems.

2.3 Related Work

There are a number of synthetic benchmarks investigated for studying I/O performance on HPC systems. Popular desktop system I/O benchmarks such as IOZone [13] and FileBench [15], are generally not relevant to HPC applications as they either do not have parallel implementations or exercise I/O patterns that are uncommon in scientific applications. The SPIObench [23] and PIORAW [16] are parallel benchmarks that read and write to separate files in parallel fashion, but offer no way to assess performance for concurrent accesses to a single file. The Effective I/O Benchmark [17] reports a I/O performance number by running a set of predefined configurations for a short period of time. It is thus difficult to relate its performance back to applications or compare results with other available benchmarks. The FLASH-IO [8] benchmark, studied extensively in a paper on paralleNetCDF [6], is probably closest in spirit to MADbench2 in that it was extracted directly from a production application — the FLASH code used for simulating SN1a supernovae. However, it focuses exclusively on write performance, which is only half of what we require to understand performance for data analysis applications. The LLNL IOR [12] benchmark is a fully synthetic benchmark that exercises both concurrent read/write operations and read/write operations to separate files (one-file-per-processor). IOR is highly parameterized, allowing it to mimic a wide variety of I/O patterns. However, it is difficult to relate the data collected from IOR back to the original application requirements. Some preliminary work has been done to relate IOR performance to MADbench2 [22]. A recent comparison of I/O benchmarks can be found in a study by Saini, et al. [18].

3. EXPERIMENTAL TESTBED

Our work uses MADbench2 to compare read and write performance with (i) unique and shared files, (ii) POSIX- and MPI-IO APIs, and (iii) synchronous and asynchronous MPI-IO, both within individual systems at varying concurrencies (16, 64 & 256 CPUs) and across seven leading HPC systems. These platforms include six architectural paradigms — Cray XT3, IA64 Quadrics cluster, Power5 SP, Opteron Infiniband cluster, BlueGene/L (BG/L), and SGI Altix — and four state-of-the-art parallel filesystems — Lustre [4], GPFS [21], PVFS2 [14], and CXFS [7].

CPU	BG/L CPU	NPIX	NBIN	Mem (GB)	Disk (GB)
—	16	12,500	8	6	9
16	64	25,000	8	23	37
64	256	50,000	8	93	149
256	—[2]	100,000	8	373	596

Table 1: Configurations and the associated memory and disk requirements for I/O-mode MADbench2 experiments. Due to its memory constraints BG/L uses four times the number of CPUs for a given problem size.

In order to minimize the communication overhead of *pdgemm* calls, at each concurrency we choose NPIX to fill the available memory. This also minimizes the opportunities for system level buffering to hide I/O costs. Many systems can scavenge otherwise empty memory to buffer their I/O, resulting in much higher data-rates that often exceed the system's theoretical peak performance. However, this is an unrealistic expectation for most real applications, since memory is too valuable a commodity to leave empty like this.

[1]For all the systems studied here, optimal performance was seen for RMOD = WMOD = 1, or free-for-all I/O.

[2]We were unable to conduct 1024-way BG/L experiments as the ANL runs would have required multiple days of execution, exceeding the maximum allowed by the queue schedulers.

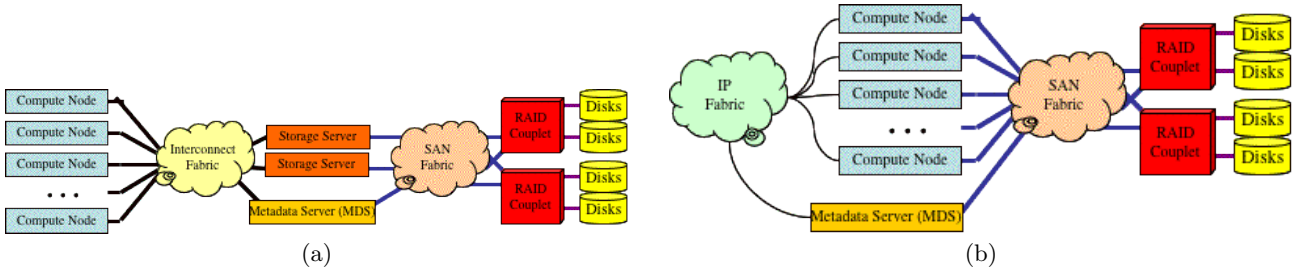

Figure 1: The general architecture of cluster filesystem we have studied, including Lustre, PVFS2, and GPFS, follow the abstract pattern shown in (a); the CXFS configuration is more similar to a SAN, shown in (b).

Setting aside about one quarter of the available memory for other requirements, we choose NPIX such that $5 \times \texttt{NPIX}^2 \times 8 \sim 3/4 \times \texttt{M} \times \texttt{P}$ for P processors each having M bytes of memory. As shown in Table 1, this approach implies weak scaling with the number of pixels doubling (and the size of the matrices quadrupling) between the chosen concurrencies, while the size of the data read/written in each I/O call from each processor is constant (78 MB/CPU/call for the BG/Ls, 312 MB/CPU/call otherwise) across all concurrencies.

The systems considered here typically have 2 GB/CPU, of which our chosen problem sizes filled 73%. The BG/L systems have only 0.5 GB/CPU so the processor counts were quadrupled for a given problem size, again filling 73% of the available memory; this allows us to use the same runs to compare performance at both a fixed concurrency and a fixed problem size. The Power5 SP system has 4 GB/CPU; in order to support the same set of concurrencies and problem sizes we therefore had to relax the dense data requirement, and ran each of the standard problem sizes on the smallest standard concurrency that would support it. Since this only filled 36% of the available memory we also ran tests with dense data (using 35K, 70K and 140K pixels on 16, 64 and 256 CPUs respectively, filling 71% of the available memory in each case) to ensure that this system's performance was not being exaggerated by buffering, and found that the sparse and dense data results at a given concurrency never differed by more than 20%.

3.1 Overview of Parallel Filesystems

In this section, we briefly outline the I/O configuration of each evaluated supercomputing systems, a summary of architecture and filesystem features are shown in Table 2 and Figure 1.

The demands of coordinating parallel access to filesystems from massively parallel clusters has led to a more complex system architecture than the traditional client-server model of NFS. A number of HPC-oriented Storage Area Networks (SAN), including GPFS, Lustre, CXFS, and PVFS2 address some of the extreme requirements of modern HPC applications using a hierarchical multi-component architecture. Although the nomenclature for the components of these cluster filesystems differs, the elements of each architecture are quite similar. In general, the requests for I/O from the compute nodes of the cluster system are aggregated by a smaller set of nodes that act as dedicated I/O servers.

In terms of filesystem metadata (i.e. file and directory creation, destruction, open/close, status and permissions), Lustre and GPFS operations are segregated from the bulk I/O requests and handled by nodes that are assigned as dedicated metadata servers (MDS). The MDS process file operations such as open/close, which can be distinct from the I/O servers that handle the bulk read/write requests. In the case of PVFS2, the metadata can be either segregated (just as with GPFS or Lustre) or distributed across the I/O nodes, with no such segregation of servers.

The bulk I/O servers are referred to as Object Storage Servers (OSS) in Lustre nomenclature whereas they are Network Shared Disk (NSD) in GPFS nomenclature for network shared disk and Virtual Shared Disk (VSD) for the storage server model of operation, and "data servers" in PVFS nomenclature. The back-end storage devices for Lustre are referred to as Object Storage Targets (OSTs) whereas PVFS2 and GPFS rely on either locally attached storage or a SAN for the back-end storage devices.

The front-end of the I/O servers (both bulk I/O and metadata) are usually connected to the compute nodes via the same interconnection fabric that is used for message passing between the compute nodes. One exception to this is the BG/L system, which has a dedicated Tree/Collective network links to connect the compute nodes to I/O nodes within their partition. The BG/L I/O nodes act as the storage clients and connect to the I/O servers via a GigE network. Another exception is CXFS where the storage is directly attached to the nodes using the Fiberchannel (FC) SAN fabric and coordinated using a dedicated metadata server that is attached via Ethernet. The back-end of the I/O servers connect to the disk subsystem via a different fabric — often FC, Ethernet or Infiniband, but occasionally locally attached storage such as SATA or SCSI disks that are contained within the I/O server.

3.2 Jaguar: Lustre on Cray XT3

Jaguar, a 5,200 node Cray XT3 supercomputer, is located at Oak Ridge National Laboratory (ORNL) and uses the Lustre parallel filesystem. Each XT3 node contains a dual-core 2.6 GHz AMD Opteron processor, tightly-integrated to the XT3 interconnect via a Cray SeaStar-1 ASIC through a 6.4 GB/s bidirectional HyperTransport interface. All the SeaStar routing chips are interconnected in a 3D torus topology, where each node has a direct link its six nearest neighbors. The XT3 uses two different operating systems: Catamount 1.4.22 on compute processing elements (PEs) and Linux 2.6.5 on service PEs.

Jaguar uses 48 I/O OSS servers, and one MDS, connected via the custom SeaStar-1 network in a toroidal configuration to the compute nodes. The OSSs are connected to a total of 96 OSTs that use Data Direct Networks (DDN) 8500 RAID controllers as backend block devices for the OSTs. There are 96 OSTs, 8 per DDN couplet, and nine approximately 2.5 GB/s (3 GB/s peak) DDN8500 couplets — providing a

Name Machine	Parallel File System	Proc Arch	Interconnect Compute to I/O nodes	Max Node BW to I/O	Measured Node BW	I/O Servers/ Client	Max Disk BW	Total Disk (TB)
Jaguar	Lustre	Opteron	SeaStar-1	6.4	1.2	1:105	22.5	100
Thunder	Lustre	Itanium2	Quadrics	0.9	0.4	1:64	6.4	185
Bassi	GPFS	Power5	Federation	8.0	6.1	1:16	6.4	100
Jacquard	GPFS	Opteron	InfiniBand	2.0	1.2	1:22	6.4	30
SDSC BG/L	GPFS	BG/L	GigE	0.2	0.18	1:8	8	220
ANL BG/L	PVFS2	BG/L	GigE	0.2	0.18	1:32	1.3	7
Columbia	CXFS	Itanium2	FC4	1.6	—[4]	—[4]	1.6[5]	600

Table 2: Highlights of architectures and file systems for examined platforms. All bandwidths (BW) are in GB/s. The "Measured Node BW" uses MPI benchmarks to exercise the fabric between nodes.

theoretical 22.5 GB/s aggregate *peak* I/O rate. The maximum sustained performance, as measured by ORNL's system acceptance tests, was 75%-80% of this theoretical peak for compute clients at the Lustre FS level[3].

3.3 Thunder: Lustre on IA64/Quadrics

The Thunder system, located at Lawrence Livermore National Laboratory (LLNL), consists of 1024 nodes, and also uses the Lustre parallel filesystem. The Thunder nodes each contain four 1.4 GHz Itanium2 processors running Linux Chaos 2.0, and are interconnected using Quadrics Elan4 in a fat-tree configuration. Thunder's 16 I/O OSS servers (or GateWay (GW) nodes) are connected to the compute nodes via the Quadrics interconnection fabric and deliver ∼400MB/s of bandwidth each, for a total of 6.4 GB/s peak aggregate bandwidth. Two metadata servers (MDS) are also connected to the Quadrics fabric, where the second server is used for redundancy rather than to improve performance. Each of the 16 OSS's are connected via 4 bonded GigE interfaces to a common GigE switch that in turn connects to a total of 32 OSTs, delivering an aggregate peak theoretical performance of 6.4 GB/s.

3.4 Bassi: GPFS on Power5/Federation

The 122-node Power5-based Bassi system is located at Lawrence Berkeley National Laboratory (LBNL) and employs GPFS as the global filesystem. Each node consists of 8-way 1.9 GHz Power5 processors, interconnected via the IBM HPS Federation switch at 4 GB/s peak (per node) bandwidth. The experiments conducted for our study were run under AIX 5.3 with GPFS v2.3.0.18.

Bassi has 6 VSD servers, each providing sixteen 2 Gb/s FC links. The disk subsystem consists of 24 IBM DS4300 storage systems, each with forty-two 146 GB drives configured as 8 RAID-5 (4data+1parity) arrays, with 2 hot spares per DS4300. For fault tolerance, the DS4300 has dual controllers; each controller has dual FC ports. Bassi's maximum theoretical I/O bandwidth is 6.4 GB/s while the measured aggregate disk throughput of our evaluated configuration is 4.4 GB/s read and 4.1 GB/s write using the NERSC PIORAW benchmark.[6]

3.5 Jacquard: GPFS on Opteron/Infiniband

The Jacquard system is also located at LBNL and contains 320 dual-socket (single-core) Opteron nodes running Linux 2.6.5 (PathScale 2.0 compiler). Each node contains two 2.2 GHz Opteron processors interconnected via InfiniBand (IB) fabric in a 2-layer fat-tree configuration, where IB 4x and 12x are used for the leaves and spines (respectively) of the switch. Jacquard uses a GPFS-based parallel filesystem that employs 16 (out of a total of 20) I/O servers for scratch connected to the compute nodes via the IB fabric. Each I/O node has one dual-port 2 Gb/s QLogic FC card, which can deliver 400 MB/s, hooked to a S2A8500 DDN controller. The DDNs serving scratch (the filesystem examined in our study) has a theoretical peak aggregate performance of 12 GB/s, while the compute nodes have a theoretical peak bandwidth of 6.4 GB/s. However, peak bandwidth is limited since the IP over IB implementation sustains only ∼200-270 MB/s, thereby limiting the aggregate peak to ∼4.5 GB/s. The maximum measured aggregate I/O bandwidth available on Jacquard is 4.2 GB/s throughput (reads and writes) using NERSC's PIORAW benchmark.

3.6 SDSC BG/L: GPFS on BlueGene/L

The BG/L at the San Diego Supercomputing Center consists of 1024-nodes each containing two 700MHz PowerPC 440 processors, on-chip memory controller and communication logic, and 512MB of memory. The BG/L nodes are connected via three independent networks: the IBM BlueGene Torus, the Global Tree, and the Global Interrupt. I/O aggregation on the BG/L platform is accomplished using the Tree network, which also services collective operations.

BG/L systems dedicate a subset nodes to forward I/O requests (and other system calls) trapped by the compute node kernel (CNK) on the compute nodes in the system and forward those requests via a Gigabit Ethernet to the data servers. These I/O nodes are only visible to the compute nodes within their partition. Whereas the compute nodes on BG/L generally run the IBM CNK microkernel, the I/O nodes run a full fledged Linux kernel. Any system calls executed on the compute nodes are trapped on the compute nodes by CNK and then forwarded to the appropriate I/O node to be re-executed as a real system call with the appropriate user ID. The I/O nodes can act as clients for a number of different cluster filesystem implementations including GPFS, NFS, Lustre, and PVFS. The ratio of I/O nodes to compute nodes is a system-dependent parameter.

The SDSC platform employees GPFS and uses the most aggressive I/O system configuration possible for a BG/L, offering a 1:8 ratio of I/O servers to compute nodes. The

[3]Personal communication Jaguar system administrator.
[4]Inapplicable to Columbia's SAN implementation.
[5]Although 3.0+ GB/s are available across the system, only 1.6 is available to each 512-way SMP.
[6]See `https://www.nersc.gov/nusers/resources/bassi/perf.php\#bmresults`.

I/O servers forward request via Gigabit Ethernet to a second tier of IA64-based NSD servers that connect to the back-end storage fabric. There are two filesystems visible from the compute nodes, both of which are evaluated in our study. The older scratch filesystem uses 12 NSD and 2 MDS servers, while the newer higher performance WAN filesystem uses 58 NSD and 6 MDS servers.

3.7 ANL BG/L: PVFS2 on BlueGene/L

Like the SDSC system, the ANL BG/L system contains 1024 compute nodes, however the ANL system uses the PVFS2 filesystem and employs only 32 I/O nodes (a ratio of 1:32 I/O nodes per compute node). The I/O nodes act as storage "clients" for the PVFS2 filesystem. The I/O nodes connect via a GigE switch to 16 dual-processor Xeon-based "storage" nodes. The storage nodes serve up 6.7 TB of locally attached disk that is are configured as a RAID5 using ServeRAID6i+ integral disk controllers. Assuming 900 MB/s for each storage controller network link, the PFVS2 filesystem can provide a peak aggregate I/O bandwidth of 1.3 GB/s read or write.

The two BG/L's represent the most similar pair of systems in this study, allowing the closest direct comparison between filesystem instantiations. However, there are a number of key implementation details that differ across these two systems, particularly the filesystem implementations (GPFS and PVFS2 respectively) and the varying ratios of I/O server nodes to compute nodes (1:8 and 1:32 respectively). These factors need to be taken into consideration in the comparative analysis of their performance results.

3.8 Columbia: CXFS on SGI Altix3700

Columbia is a 10,240-processor supercomputer which is located at NASA Ames Research Center and employed CXFS. The Columbia platform is a supercluster comprised of 20 SGI Altix3700 nodes, each containing 512 1.5 GHz Intel Itanium processors and running Linux 2.6.5. The processors are interconnected via NUMAlink3, a custom network in a fat-tree topology that enables the bisection bandwidth to scale linearly with the number of processors.

CXFS is SGI's parallel SAN solution that allows the clients to connect directly to the FC storage devices without an intervening storage server. However the metadata servers (MDS) are connected to the clients via GigE to each of the clients in the SAN. The SAN disk subsystem is an SGI Infinite Storage 6700 disk array that offers a peak aggregate read/write performance of 3 GB/s. Columbia is organized as three logical domains, and our experiments only use the "science domain" which connects one front-end node, 8 compute nodes and 3 MDS to the disk subsystem via a pair of 64-port FC4 (4 Gb/s FC) switches. Each of the eight compute nodes speak to the disk subsystem via 4 bonded FC4 ports. The disk subsystem itself is connected to the FC4 switches via eight FC connections, but since our compute jobs could not be scheduled across multiple Altix3700 nodes, we could use a maximum of four FC4 channels for a theoretical maximum of 1.6 GB/s per compute node.

4. SYNCHRONOUS I/O BEHAVIOR

This section presents performance on our evaluated test suite using the synchronous I/O configuration, where computation and communication is not explicitly overlapped. We conduct four experiments for each of the configurations shown in Table 1: POSIX I/O with unique files, POSIX I/O with a shared file, MPI-IO with unique files, and MPI-IO with a shared file.

Early in the experience with cluster filesystems such as NFS we found that POSIX I/O to a single shared file would result in undefined behavior or extraordinarily poor performance, thereby requiring MPI-IO to ensure correctness. Results show that our set of evaluated filesystems now ensure correct behavior for concurrent writes. Collective MPI-IO should, in theory, enable additional opportunities to coordinate I/O requests so that they are presented to the disk subsystem in optimal order. However, in practice we saw trivial differentiation between POSIX and MPI-IO performance for MADbench2's contiguous file access patterns, which do not benefit from the advanced features of MPI-IO. Indeed, due to similarity between POSIX and MPI-IO performance, we omit any further discussion between these two APIs and present the better of the two measured runtimes.

Figures 2 and 3 present summaries of synchronous MADbench2 results on the seven evaluated architectural platforms, using the default environment parameters, with performance compared via fixed concurrencies and fixed total I/O throughput (respectively). While a breakdown of each individual system is shown in Figure 4. Detailed performance analysis as well as optimized execution results are presented in the subsequent subsections.

4.1 Jaguar Performance

Figure 4(a-b) presents the synchronous performance on the Jaguar system. For unique file reading and writing, Jaguar shows a significantly higher aggregate I/O rate compared with all other evaluated architectures, as can clearly be seen in Figure 2(a-b). As with most cluster filesystems, Jaguar cannot reach peak performance at low concurrencies because the I/O rate is limited by the effective interconnect throughput of the client nodes. For instance, the effective unidirectional messaging throughput (the "injection bandwidth" as opposed to SeaStar's max throughput transit bandwidth) of each Jaguar node is 1.1 GB/s. Thus 8 nodes (16 processors) only aggregates 8.8 GB/s of bandwidth; however, as can be seen in Figure 4(a), increasing concurrency to 64 nodes causes the Jaguar I/O system to rapidly approach saturation bandwidth of its disk subsystem for reading and writing unique files. At 256 processors, the Jaguar disk subsystem is nearly at its theoretical peak bandwidth for writes to unique files. Observe that reading is consistently slower than writing for all studied concurrencies. We attribute this to I/O buffering effects. The I/O servers are able to hide some of the latency of committing data to disk when performing writes; however, reads cannot be buffered unless there is a temporal recurrence in the usage of disk blocks — thereby subjecting the I/O request to the full end-to-end latency of the disk subsystem.

In comparison to accessing unique files (one file per processor), Jaguar's performance when reading/writing to single shared files is uniformly flat and performs poorly at all concurrencies when run using the default environment settings, as can be seen in Figure 4(a). Thus, its default shared performance is relatively poor compared with Bassi, Jacquard, and the SDSC BG/L, as shown in Figure 2(c-d). The limited shared performance arises because all I/O traffic is restricted to just 8 of the 96 available OSTs in default mode. This policy ensures isolation of I/O traffic to provide more

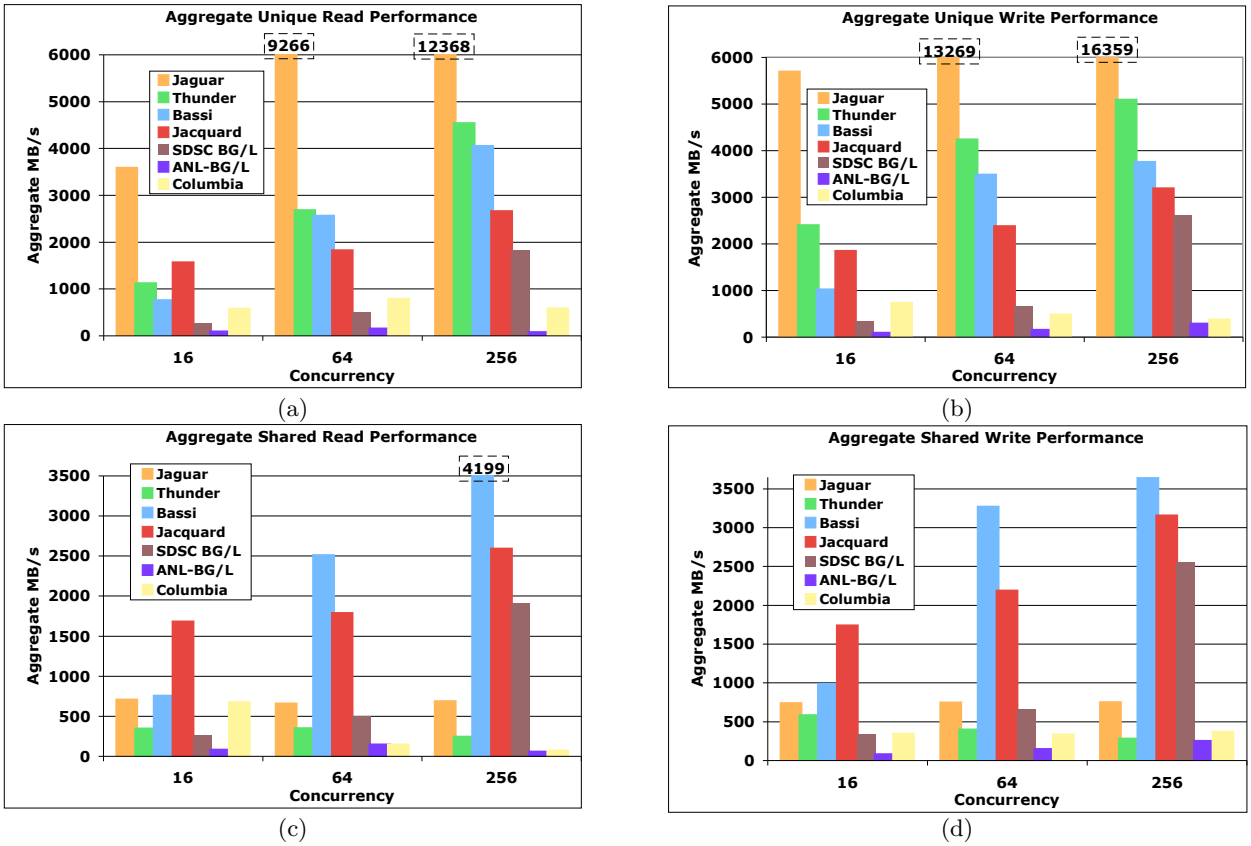

Figure 2: Synchronous MADbench2 aggregate throughput results for fixed concurrencies using the default environment parameters, showing unique file (a) read and (b) write performance, and shared file (c) read and (d) write performance. Note that the Jaguar unique file performance greatly exceeds the limits of the ordinate axes.

consistent performance to each job in a mixed/mode batch environment with many competing processes, but restricts the ability of any single job to take advantage of the full I/O throughput of the disk subsystem. Using the `lstripe` command, we were able to assign 96 OSTs to each data file, with results show in Figure 4(b). This optimized striping configuration is not the default setting because it (i) exposes the job to increased risk of failure, as the loss of any single OST will cause the job to fail, (ii) exposes the job to more interference from other I/O intensive applications, and (iii) reduces the performance of the one-file-per-processor configuration. However, for parallel I/O into a single file, the striping approach results in dramatic improvements for both the read and write performance. Unfortunately the striping optimization causes unique-file I/O to drop in performance relative to the default configuration, due to increased interference of disk operations across the OSTs.

4.2 Thunder Performance

In general, the behavior of Thunder's Lustre I/O system for the synchronous tests is remarkably similar to that of Lustre on Jaguar despite considerable differences in machine architecture. As seen in Figure 2 Thunder achieves the second highest aggregate performance after Jaguar for unique file accesses (at higher concurrencies). This is not surprising since Thunder's peak I/O performance is only one-fifth that of Jaguar's. Figure 4(c) shows that, like Jaguar, Thunder write performance is better than the read throughput since memory buffering is more beneficial for writes than for reads. Additionally, performance in shared-file mode is dramatically worse than using unique files (one-file per process). Unlike Jaguar, however, results do not show substantial performance benefits from striping using the `lstripe` command. In theory, a well tuned striping should bring the shared file performance to parity with the default environmental settings. The LLNL system administrators believe that the differences between Lustre performance on Jaguar and Thunder is largely due to the much older software and hardware implementation deployed on Thunder. Future work will examine Thunder performance under the upgraded software environment.

4.3 Bassi Performance

As seen in Figure 4(d), unlike the Lustre-based systems, Bassi's GPFS filesystem offers shared-file performance that is nearly an exact match of the unique-file behavior at each concurrency using the system's default configuration. As a result, Bassi attains the highest performance of our test suite for shared-file accesses (see Figure 2), with no special optimization or tuning. The I/O system performance ramps up to saturation rapidly, which is expected given the high bandwidth interconnect that links Bassi's compute nodes to

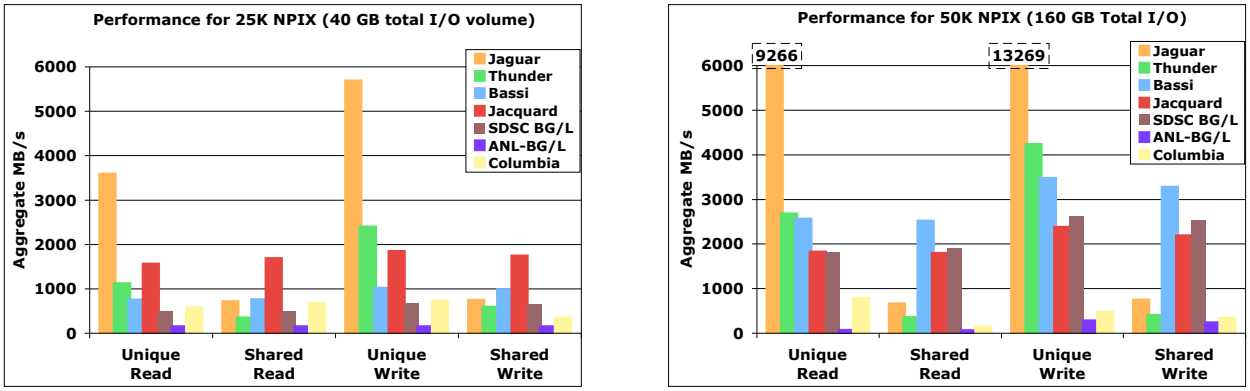

Figure 3: Comparison of performance for fixed total I/O throughput, using (a) 40GB for 25K NPIX and (b) 160GB for 50K NPIX. Note that, for a fixed number of pixels, the BG/L systems use four times as many processor as the other evaluated platforms due to memory constraints.

the I/O nodes. With 6 I/O nodes, we would expect that any job concurrency higher than 48 processors would result in I/O performance that is close to saturation. The synchronous I/O performance bears out that assumption given the fall-off in write performance for job concurrencies beyond 64 processors (8 nodes). The read performance trails the write performance for job sizes below 256 processors, and then exceeds the write performance for higher concurrencies. There is no obvious explanation for this behavior, but IBM engineers have indicated that the GPFS disk subsystem is capable of recognizing data access patterns for reads and prefetching accordingly. However, IBM offers no direct way of determining whether the pattern matching or prefetching has been invoked.

4.4 Jacquard Performance

The Jacquard platform provides an opportunity to compare and contrast GPFS performance on a system that is a different hardware architecture and OS than the AIX/PowerPC systems for which GPFS was originally developed. Figure 4(e) shows that GPFS on Linux offers similar performance characteristics to the AIX implementation on Bassi. The performance of reading and writing to a shared file matches that of writing one file per processor without requiring any specific performance tuning or use of special environment variables. Unlike Bassi, Jacquard performance continues to scale up even at a concurrency of 256 processors, indicating that the underlying disk storage system or I/O servers (GPFS NSDs) — which have a theoretical peak bandwidth of 4.5 GB/s — have probably not been saturated. However, as seen in Figure 2, Bassi generally outperforms Jacquard due to its superior node to I/O bandwidth (see Table 2).

4.5 BG/L GPFS

The BG/L platform at SDSC offers yet another system architecture and OS combination for examining GPFS performance while presenting an interesting comparison to the ANL BG/L configuration. When comparing absolute performance across architectures, it should be noted the BG/L (single rack) platforms are the two smallest systems in our study (in terms of peak Gflop/s rate). As seen in Figure 2, despite its low performance for small numbers of processors, at high concurrencies the SDSC BG/L (WAN filesystem) is quite competitive with the larger systems in this study. Additionally, if we compare data with respect to a fixed I/O throughput — as shown in Figure 3 (where BG/L uses four times as many processors) — results show that for 50K pixels, the SDSC BG/L WAN system achieves a significant performance improvement relative to the other platforms, attaining or surpassing Jacquard's throughput. This affect was much less dramatic for the 25K pixel experiment.

Figure 4(f) compares the two filesystems at SDSC, showing that the WAN configuration (right) achieves significantly higher performance than the local filesystem (left). The WAN filesystem has many more spindles attached (hence the disk subsystem has much higher available bandwidth) and in addition, it has many more NSDs attached than the local scratch. Consequently, the local filesystem reaches bandwidth saturation at a much lower concurrency than the WAN. Indeed, as seen in Figure 4(g), the 256-way WAN experiment has clearly not saturated the underlying filesystem throughput. Consistent with the other GPFS-based systems in our study, the SDSC I/O rate for concurrent reads/writes into a single shared file match the performance of reads/writes to unique files, using the default configuration with no tuning required.

4.6 BG/L PVFS2

Figure 4(h) presents the performance of the PFVS2 disk subsystem on the ANL BG/L platform, which shows relatively low I/O throughput across the various configurations. When comparing the scaling behavior between the SDSC and ANL BG/L systems (Figure 2), we observe that (using default tuning parameters) the PVFS2 filesystem sees good scalability for both the unique and shared file write performance on up to 64 processors; however, beyond that concurrency the file read performance goes down significantly. The much smaller number of I/O server nodes may have difficulty fielding the large number of simultaneous read requests.

ANL provided us with a number of tunable parameters to improve the I/O performance on the PVFS2 filesystem, including the `strip size` (equivalent to the "chunk size" in a RAID environment) and `num files` which controls how new

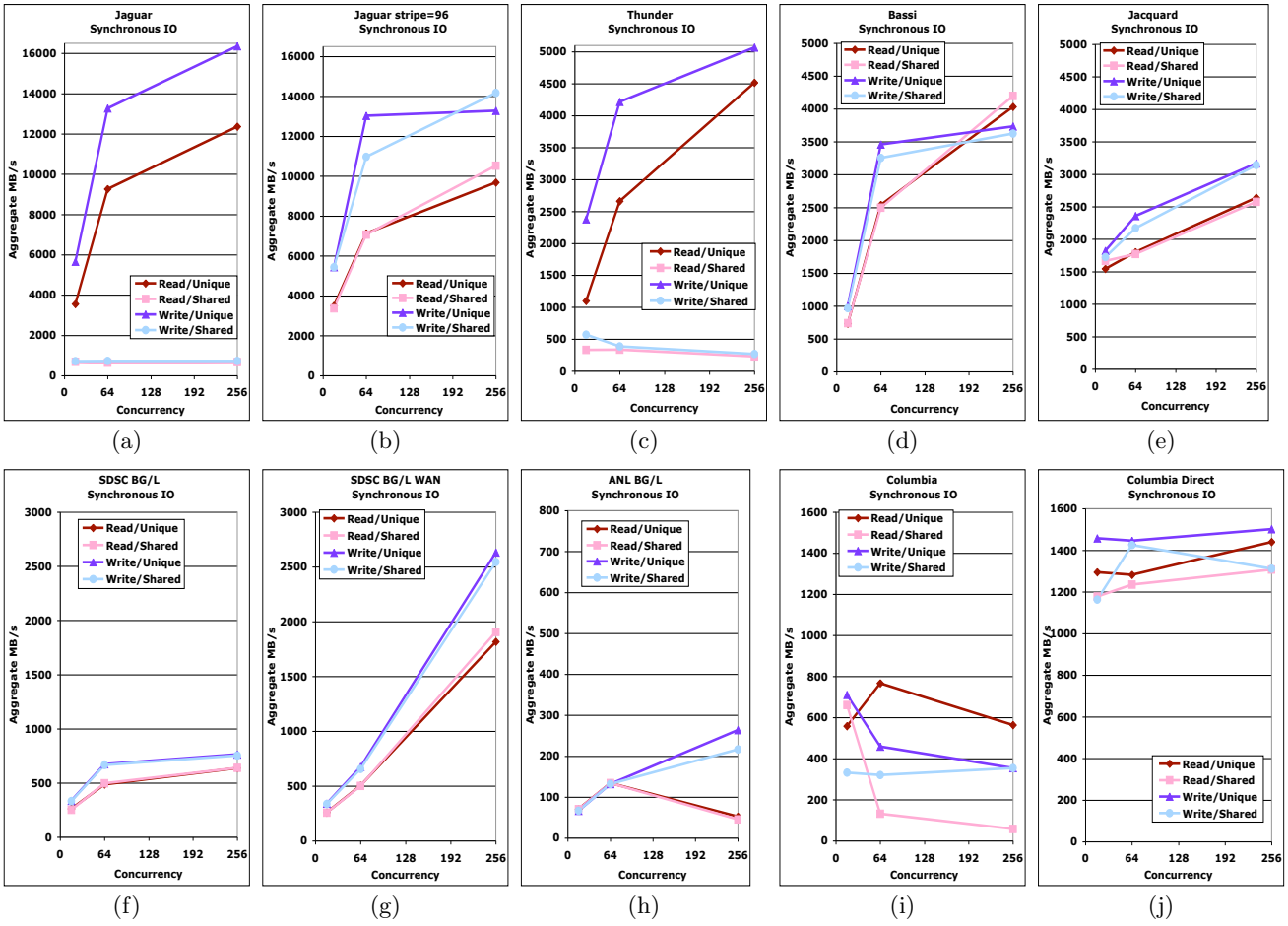

Figure 4: MADbench2 performance on each individual platforms showing (a) Jaguar default environment (b) Jaguar optimized striping configuration (c) Thunder default environment (d) Bassi default environment (e) Jacquard default environment (f) SDSC BG/L local scratch (g) SDSC BG/L WAN filesystem (h) ANL BG/L default environment (i) Columbia default environment (j) Columbia using directIO interface.

files are distributed across the multiple I/O servers. The latter parameter is analogous to the Lustre `lstripe` tunable parameter which also controls how files are striped across the Lustre I/O servers (OSTs). After performing a search across the `strip size` and `num files` parameter space and attempting experiments with smaller memory footprints we were unable to derive a significant performance benefit for MADbench2 on this platform. This also holds true when comparing data in terms of fixed total I/O throughput as shown in Figure 3. Future work will continue exploring possible optimization approaches.

One key difference between SDSC and ANL BG/L systems is the compute–I/O node ratio (8:1 and 32:1 for the SDSC and ANL systems respectively). Although the primary aim of this work is to compare filesystems as they are currently deployed, it is interesting to consider how performance would differ between the two BG/L systems if this ratio was the same on both platforms. To normalize this comparison, we ran experiments using 4x the required processes on the ANL system and mapped them in such a way that that the effective compute–I/O node ratio became 8:1 (three of the four allocated processes remain idle during computation). In the 8:1 ANL experiment — which offers a 4x increase in the default compute–I/O node ratio — the average performance of the ANL system throughput improved 2.6x. These results indicate that the smaller number of I/O nodes are indeed an important bottleneck for I/O system performance. Nonetheless, the 8:1 ANL configuration was still 4.7x slower on average than the default 8:1 SDSC WAN, confirming that ratio of I/O nodes is only one of the many factors that affects I/O throughput.

4.7 Columbia

Figure 4(i) shows the baseline performance of the Columbia CXFS filesystem. Observe that, except for the read-unique case, most of the I/O operations reach their peak performance at the lowest concurrency, and performance falls off from there. This is largely due to the fact that the I/O interfaces for the 256 processor Altix nodes are shared across the entire node, thus increased concurrency does not expose the application to additional physical data channels to the disk subsystem. Therefore, Columbia tends to reach peak performance at comparatively low concurrency compared to the other systems in our study. As processor count increases, there is increased lock overhead to coordinate the access to the Unix block buffer cache, more contention for the physi-

cal interfaces to the I/O subsystem, and potentially reduced coherence of requests presented to the disk subsystem. Overall, default Columbia I/O performance is relatively poor as seen in Figure 2.

One source of potential overhead and variability in CXFS disk performance comes from the need to copy data through an intermediate memory buffer: the Unix block buffer cache. The optimized graph in Figure 4(j) shows performance that can be achieved using the *directIO* interface. DirectIO bypasses the block-buffer cache and presents read and write requests directly to the disk subsystem from user memory. In doing so, it eliminates the overhead of multiple memory copies and mutex lock overhead associated with coordinating parallel access to the block-buffer cache. However, this approach also prevents the I/O subsystem from using the block buffer cache to accelerate temporal recurrences in the data access patterns that might be cache-resident. Additionally, directIO makes the I/O more complicated since it requires each disk transaction be block-aligned on disk, has restrictions on memory alignment, and forces the programmer to think in terms of disk-block-sized I/O operations rather than the arbitrary read/write operation sizes afforded by the more familiar POSIX I/O interfaces.

Results show that using directIO causes raw performance to improve dramatically, given the reduced overhead of the I/O requests. However, the performance peaks at the lowest processor count, indicating that the I/O subsystem (FC4) can be saturated at low concurrencies. This can actually be a beneficial feature since codes do not have to rely on high concurrencies to access the disk I/O subsystem fully. However, Columbia's peak performance as a function of the compute rate is much lower than desirable at the higher concurrencies.

5. ASYNCHRONOUS PERFORMANCE

Almost all of the systems investigated in our study show non-linear scaling in their synchronous I/O performance, indicating that some component(s) of their I/O subsystems are already beginning to saturate at only 256 processors. This is clearly a concern for ultrascale I/O-intensive applications, which will require scaling into tens or even hundreds of thousands of processors to attain petascale performance. Moreover, one of the strongest candidates for near-term petascale, the BG/L system, shows rather poor I/O performance under both GPFS and PVFS2 even at relatively low concurrencies — and cannot even run the largest data volume used in our experiments in an allowable wallclock time.

One possible way to ameliorate this potential bottleneck is to hide I/O cost behind simultaneous calculations by using asynchronous I/O facilities. Since it is often the case that I/O transfers could be overlapped with computational activity, this methodology is widely applicable to a wide range of applications. MADbench2 was therefore designed with the option of performing the simulation in an asynchronous fashion, using portable MPI-2 [9] semantics. The MPI-2 standard includes non-blocking I/O operations (the I/O calls return immediately) whose particular implementation may provide fully asynchronous I/O (the I/O performed in the background while other useful work is being done). Unfortunately only two of the seven systems under investigation — Bassi and Columbia — have MPI-2 implementations that actually support fully asynchronous I/O. On all the other systems MADbench2 successfully ran in asynchronous mode but showed no evidence of effectively backgrounded I/O.

To quantify the balance between computational rate and asynchronous I/O throughput, we developed the busy-work exponent α; recall from Section 2 that α corresponds to N^{α} flops for every N data. Thus, if the data being written/read are considered to be matrices (as in the parent MADCAP [2] application) then α=1.0 correspond to BLAS2 (matrix-vector), while α=1.5 corresponds to BLAS3 (matrix-matrix); if the same data are considered to be vectors then α=1.0 would correspond to BLAS1 (vector-vector). To evaluate a spectrum of potential numerical algorithms with differing computational intensities, we conduct 16- and 256-way experiments (with unique files) varying the busy-work exponent α from 1 to 1.5 on the two system with fully asynchronous I/O facilities.

Given MADbench2's pattern of loops over I/O and computational overhead it is expected that, given sufficient foreground work, the simulation should be able to background all but the first data read or the last data write, reducing the I/O cost by a factor of `NBIN`, which is equal to 8 for our experimental setup (see Section 3). In large-scale CMB experiments, `NBIN` is dramatically larger, allowing the potential for even more significant reductions in I/O overheads. Figure 5 presents the effective aggregate asynchronous I/O rate, as function of α, and compares it with the projected maximum performance based on the synchronous I/O throughput (for the same architecture, concurrency and run parameters).

Results clearly demonstrate the enormous potential that could be gained through an effective asynchronous I/O implementation. With high α, Bassi asynchronous I/O throughput (Figure 5(a-b)) outperforms the other systems investigated in our study (for both read and write, at all concurrencies), even gaining an advantage over Jaguar's impressive synchronous I/O throughput by about 2x at 256 processors. Similarly, Columbia's asynchronous I/O (Figure 5(c-d)) improves on its synchronous performance by a significant factor of 8x, achieving an effective aggregate 2 — 4 GB/s for reading and writing across all concurrencies; however, Columbia's failure to scale I/O indicates that the filesystem may become a bottleneck for large scale I/O-intensive applications.

Broadly speaking, the asynchronous I/O performance behaves as expected; low α reproduces the synchronous behavior, while high α shows significant performance improvements (typically) approaching the projected throughput. Notice that the critical value for the transition being somewhere between 1.3 and 1.4, corresponding to an algorithmic scaling $> \mathcal{O}(N^{2.6})$ for matrix computations. This implies that (for matrices) only BLAS3 calculations will have sufficient computational intensity to hide their I/O effectively. An open question is how α might change with future petascale architectures; if computational capacities grow faster than I/O rates then we would expect the associated α to grow accordingly. Given how close current system balance is to the practical limit of BLAS3 complexity, this is an issue of some concern.

6. SUMMARY AND CONCLUSIONS

In this paper, we examine the I/O components of several state-of-the-art parallel architectures — an aspect that is becoming increasingly critical in many scientific domains due to the exponential growth of sensor and simulated data vol-

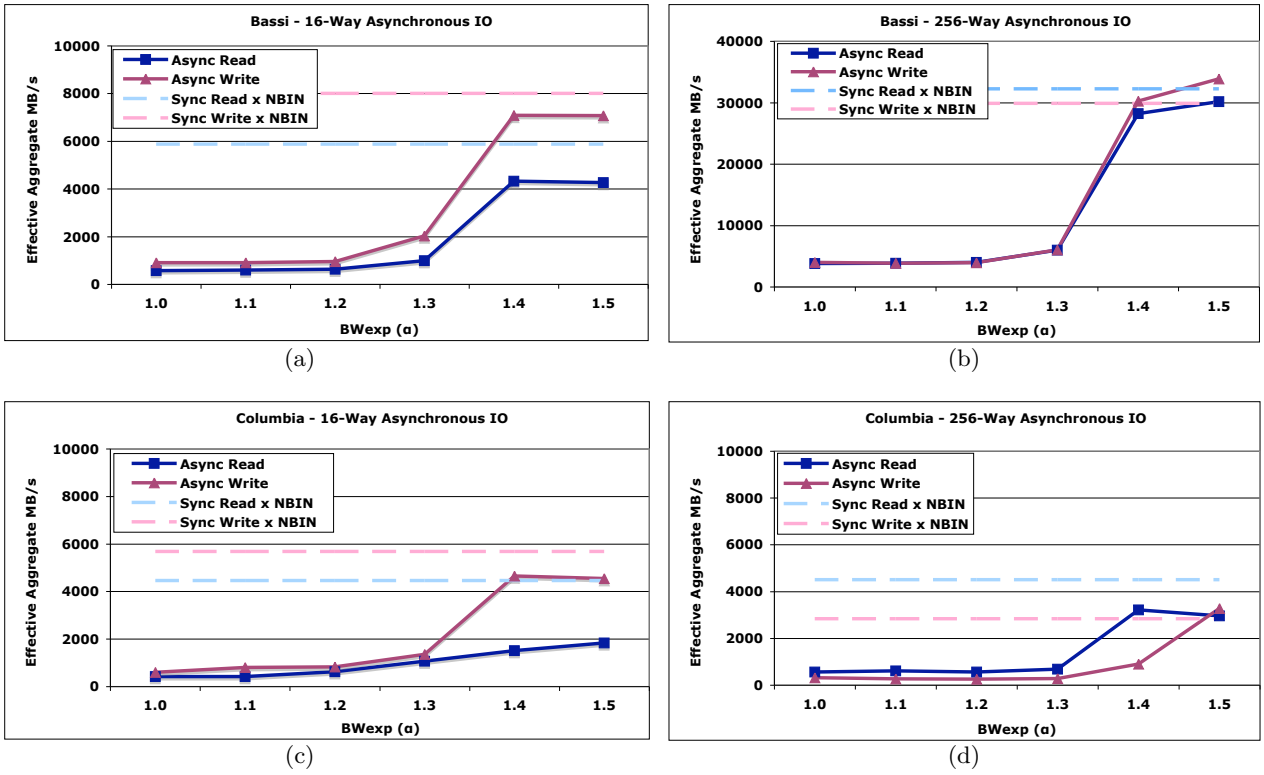

Figure 5: 16-way and 256-way asynchronous I/O performance as a function of the busy-work exponent, for the two evaluated platforms supporting this functionality (a-b) Bassi and (c-d) Columbia. The busy-work exponent (α) of 1.0 and 1.5 correspond to BLAS2 (matrix-vector) and BLAS3 (matrix-matrix) computational intensities (respectively).

umes. Our study represents one of the most extensive I/O analyses of modern parallel filesystems, examining a broad range of system architectures and configurations. Note that although cost is, of course, a critical factor in system acquisition, our evaluation is purely performance based and does not take global filesystem costs into consideration.

We introduce MADbench2, a second-generation parallel I/O benchmark that is derived directly from a large-scale CMB data analysis package. MADbench2 is lightweight and portable, and has been generalized so that (in I/O mode) the computation per I/O-datum can be set by hand, allowing the code to mimic any degree of computational complexity. This benchmark therefore enables a more concrete definition of the appropriate balance between I/O throughput and delivered computational performance in future systems for I/O intensive data analysis workloads.

In addition, MADbench2 allows us to explore multiple I/O strategies and combinations thereof, including POSIX, MPI-IO, shared-file, one-file-per-processor, and asynchronous I/O, in order to find the approach that is best matched to the performance characteristics of the underlying systems. Such a software design-space exploration would not be possible without use of a scientific-application derived benchmark.

Results show that, although concurrent accesses to shared files in the past required MPI-IO for correctness on clusters, the modern filesystems evaluated in this study are able to ensure correct operation even with concurrent access using the POSIX APIs. Furthermore, we found that there was virtually no difference in performance between POSIX and MPI-IO performance for the large contiguous I/O access patterns of MADbench2.

Contrary to conventional wisdom, we have also found that — with properly tuned modern parallel filesystems — you can (and should) expect filesystem performance for concurrent accesses to a single shared file to match the performance when writing into separate files! This can be seen in the GPFS and PVFS2 filesystems, which provide nearly identical performance between the shared and unique files using default configurations. Results also show that, although Lustre performance for shared files is inadequate using the default configuration, the performance difference can be fixed trivially using the `lstripe` utility. CXFS provides broadly comparable performance for concurrent single-file writes, but not reads.

Additionally, our experiments show that the distributed-memory systems required moderate numbers of nodes to aggregate enough interconnect links to saturate the underlying disk subsystem. Failure to achieve linear scaling in I/O performance at concurrencies beyond the point of disk subsystem saturation was not expected, nor was it observed. The number of nodes required to achieve saturation varied depending on the balance of interconnect performance to peak I/O subsystem throughput. For example a Columbia node required less than 16 processors to saturate the disks, while the SDSC BG/L system did not reach saturation even at the maximum tested concurrency of 256 processors.

Finally, we found that MADbench2 was able derive enormous filesystem performance benefits from MPI-2's asynchronous MPI-IO. Unfortunately, only two of our seven evaluated systems, Bassi and Columbia, supported this functionality. Given enough calculation, asynchronous I/O was able to improve the *effective* aggregate I/O performance significantly by hiding the cost of all but the first read or last write in a sequence. This allows systems with relatively low I/O throughput to achieve competitive performance with high-end I/O platforms that lack asynchronous capabilities.

To quantify the amount of comptation per I/O datum, we introduced a busy-work parameter α, which enables us to determine the minimum computational complexity that is sufficient to hide the bulk of the I/O overhead on a given system. Experimental results show that the computational intensity required to hide I/O effectively is already close to the practical limit of BLAS3 calculations — a concerning issue that must be properly addressed when designing the I/O balance of next-generation petascale systems.

Future work will continue investigating performance behavior on leading architectural platforms as well as exploring potential I/O optimization strategies. Additionally, we plan to include inter-processor communication in our analysis, which will on one hand increase the time available to hide the I/O overhead, while on the other possibly increase network contention (if the I/O and communication subsystem share common components). Finally, we plan to conduct a statistical analysis of filesystem performance to quantify I/O variability under differing conditions.

Acknowledgments

The authors would like to gratefully thank the following individuals for their kind assistance in helping us understand and optimize evaluated filesystem performance: Tina Butler and Jay Srinivasan of LBNL; Richard Hedges of LLNL; Nick Wright of SDSC; Susan Coughlan, Robert Latham, Rob Ross, and Andrew Cherry of ANL; Robert Hood and Ken Taylor of NASA-Ames. All authors from LBNL were supported by the Office of Advanced Scientific Computing Research in the Department of Energy Office of Science under contract number DE-AC02-05CH11231.

7. REFERENCES

[1] K. Antypas, A.C.Calder, A. Dubey, R. Fisher, M.K. Ganapathy, J.B. Gallagher, L.B. Reid, K. Reid, K. Riley, D. Sheeler, and N. Taylor. Scientific applications on the massively parallel bg/l machines. In *PDPTA 2006*, pages 292–298, 2006.

[2] J. Borrill. MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package. In *5th European SGI/Cray MPP Workshop*, Bologna, Italy, 1999.

[3] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas. Integrated performance monitoring of a cosmology application on leading hec platforms. In *ICPP: International Conference on Parallel Processing*, Oslo, Norway, 2005.

[4] P. Braam. File systems for clusters from a protocol perspective. In *Proceedings of the Second Extreme Linux Topics Workshop*, Monterey, CA, June 1999.

[5] J. Carter, J. Borrill, and L. Oliker. Performance characteristics of a cosmology package on leading HPC architectures. In *HiPC:International Conference on High Performance Computing*, Bangalore, India, 2004.

[6] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Efficient structured data access in parallel file systems. In *Cluster 2003 Conference*, Dec 4, 2003.

[7] D. Duffy, N. Acks, V. Noga, T. Schardt, J. Gary, B. Fink, B. Kobler, M. Donovan, J. McElvaney, and K. Kamischke. Beyond the storage area network: Data intensive computing in a distributed environment. In *Conference on Mass Storage Systems and Technologies*, Monterey, CA, April 11-14 2005.

[8] FLASH-IO Benchmark on NERSC Platforms. `http://pdsi.nersc.gov/IOBenchmark/FLASH_IOBenchmark.pdf`.

[9] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2: Advanced Features of the Message Passing Interface.* MIT Press, 1984.

[10] HECRTF:Workshop on the Roadmap for the Revitalization of High-End Computing. `http://www.cra.org/Activities/workshops/nitrd/`.

[11] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M.J.West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51-81, Feb,1988.

[12] The ASCI I/O stress benchmark. `http://www.llnl.gov/asci/purple/benchmarks/limited/ior/`.

[13] Iozone filesystem benchmark. `http://www.iozone.org`.

[14] R. Latham, N. Miller, R. Ross, and P. Carns. A next-generation parallel file system for linux clusters: An introduction to the second parallel virtual file system. *Linux Journal*, pages 56–59, January 2004.

[15] R. McDougall and J. Mauro. FileBench. `http://www.solarisinternals.com/si/tools/filebench`.

[16] The PIORAW Test. `http://cbcg.lbl.gov/nusers/systems/bassi/code_profiles.php`.

[17] R. Rabenseifner and A. E. Koniges. Effective file-I/O bandwidth benchmark. In *European Conference on Parallel Processing (Euro-Par)*, Aug 29–Sep 1, 2000.

[18] S. Saini, D. Talcott, R. Thakur, P. Adamidis, R. Rabenseifner, and R. Ciotti. Parallel I/O performance characterization of Columbia and NEC SX-8 superclusters. In *"International Parallel and Distributed Processing Symposium (IPDPS)"*, Long Beach, CA, USA, March 26-30, 2007.

[19] ScaLAPACK home page. `http://www.netlib.org/scalapack/scalapack_home.html`.

[20] SCALES: Science Case for Large-scale Simulation. `http://www.pnl.gov/scales/`.

[21] F. Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *First USENIX Conference on File and Storage Technologies Fast02*, Monterey, CA, January 2002.

[22] H. Shan and J. Shalf. Using IOR to analyze the I/O performance of HPC platforms. In *Cray Users Group Meeting (CUG) 2007*, Seattle, Washington, May 7-10, 2007.

[23] SPIOBENCH: Streaming Parallel I/O Benchmark. `http://www.nsf.gov/pubs/2006/nsf0605/spiobench.tar.gz`, 2005.