HPC Global File System Performance Analysis Using A Scientific-Application Derived Benchmark

Julian Borrill, Leonid Oliker, John Shalf, Hongzhang Shan, Andrew Uselton
CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Abstract

With the exponential growth of high-fidelity sensor and simulated data, the scientific community is increasingly reliant on ultrascale HPC resources to handle its data analysis requirements. However, to use 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 file systems, examining a broad range of system architectures and configurations, including Lustre on the Cray XT3, XT4, and Intel Itanium2 clusters; GPFS on IBM Power5 and AMD Opteron platforms; a BlueGene/P installation using GPFS and PVFS2 file systems; 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 show that only the two of the nine evaluated systems benefited from MPI-2's asynchronous MPI-IO. On those systems, experimental results indicate that the computational intensity required to hide I/O effectively is already close to the practical limit of BLAS3 calculations. Overall, our study quantifies vast differences in performance and functionality of parallel file systems across state-of-the-art platforms — showing I/O rates that vary up to 75x on the examined architectures — while providing system designers and computational scientists a lightweight tool for conducting further analysis.

1 Introduction

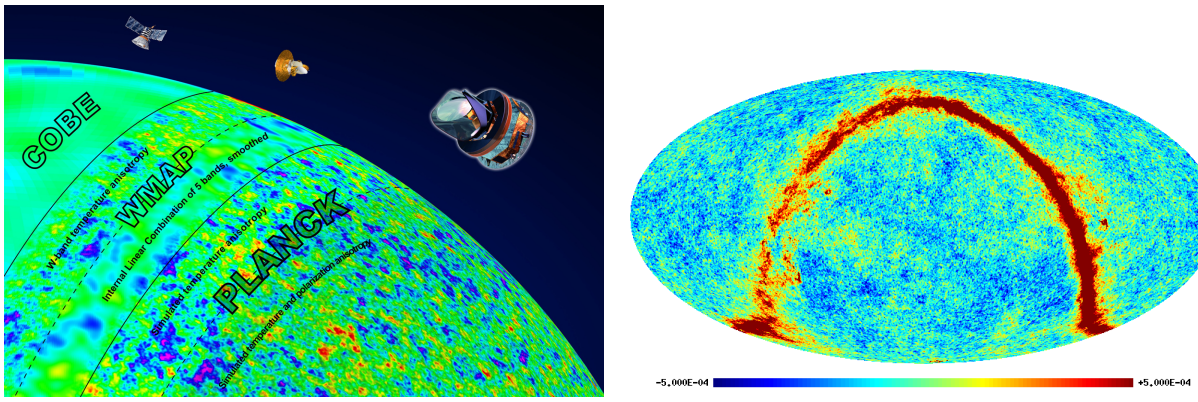
As the field of scientific computing matures, the demands for computational resources are growing rapidly. By the end of this decade numerous mission-critical applications will have computational requirements that are at least two orders of magnitude larger than current levels [11,23]. To address these ambitious goals the high-performance computing (HPC) community is racing towards making petascale computing a reality. However, to use such extreme computing power effectively, all system components must be designed in a balanced fashion to match the resource requirements of the underlying application effectively, as any architectural bottleneck will quickly render the platform intolerably inefficient. We address the I/O component of the system architecture, because in many scientific domains the volume of high-fidelity sensor and simulated data is growing exponentially.

In this paper we examine I/O performance behavior across several leading supercomputing systems and a comprehensive set of parallel file systems using a lightweight, portable benchmark called MADbench2. Unlike most of the I/O microbenchmarks currently available [12,14,15,18,20], MADbench2 is 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. Section 3 outlines the experimental testbed, as well as the extensive set of studied HPC platforms and underlying parallel file systems, including: Lustre on the Cray XT3, XT4, and Intel Itanium2 cluster; GPFS on IBM Power5 and AMD Opteron platforms; a BlueGene/P installation using GPFS and PVFS2 file systems; and CXFS on the SGI Altix3700. We 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 file system-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 performance and functionality of parallel file systems across state-of-the-art platforms, while providing system designers and computational scientists a lightweight tool for conducting further analyses (publicly available at [17]).

2 MADbench2



(a) **(b)**

Figure 1: The quest for ever higher resolution observations of ever fainter modes drives the CMB data challenge. The current state-of-the-art is the Planck satellite, (a) compared with the lower resolution COBE and WMAP satellite missions, (b) whose analysis will involve making maps with 10^8 pixels on the sky from 10^{12} observations.

MADbench2 is the second generation of a HPC benchmarking tool [3,4,6] that is derived from the analysis of massive Cosmic Microwave Background (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 rewarded with Nobel prizes in physics for its first detection of the CMB (1978: Penzias & Wilson), and for the first detection of its 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. As illustrated in Figure 1, obtaining sufficient signal-to-noise at high enough resolution requires the gathering — and then the analysis — of extremely large datasets. High performance computing has 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 MADCAP

The Microwave Anisotropy Dataset Computational Analysis Package (MADCAP) is for the analysis of CMB data and specifically for the most challenging analyses on the largest HPC systems [2]. The MADCAP

CMB angular power spectrum estimator uses Newton-Raphson iteration to locate the peak of the spectral likelihood function. This involves calculating the first two derivatives of the likelihood function with respect to the power spectrum coefficients C_l (since the CMB is azimuthally symmetric, the full spherical harmonic basis can be reduced to include the l-modes only). For a noisy pixelized CMB sky map $d_p = s_p + n_p$ (data is signal plus noise) and pixel-pixel correlation matrix $D_{pp'} = S_{pp'}(C_l) + N_{pp'}$, the log-likelihood that the observed data comes from some underlying set of C_l is

$$\mathcal{L}(d_p|C_l) = -\frac{1}{2}\left(d^T D^{-1} d + \mathrm{Tr}[\ln D]\right)$$

whose first two derivatives with respect to the C_l are

$$\frac{\partial \mathcal{L}}{\partial C_l}\frac{1}{2}\left(d^T D^{-1}\frac{\partial S}{\partial C_l}D^{-1}d - \mathrm{Tr}\left[D^{-1}\frac{\partial S}{\partial C_l}\right]\right)\frac{\partial^2 \mathcal{L}}{\partial C_l \partial C_{l'}} - d^T D^{-1}\frac{\partial S}{\partial C_l}D^{-1}\frac{\partial S}{\partial C_{l'}}D^{-1}d + \frac{1}{2}\mathrm{Tr}\left[D^{-1}\frac{\partial S}{\partial C_l}D^{-1}\frac{\partial S}{\partial C_{l'}}\right]$$

yielding a Newton-Raphson correction to an initial guess of the spectral coefficients

$$\delta C_l = \left[\frac{\partial^2 \mathcal{L}}{\partial C_l \partial C_{l'}}\right]^{-1}\frac{\partial \mathcal{L}}{\partial C_{l'}}$$

Typically, the data has insufficient sky coverage and/or resolution to obtain each multipole coefficient individually, so we bin them instead, replacing C_l with C_b.

Given the size of the pixel-pixel correlation matrices, it is important to minimize the number of matrices simultaneously held in memory; the MADCAP implementation is constrained to have no more than three matrices in core at any one time. It's operational phases, and their scalings for a map with `NPIX` pixels and `NMPL` multipoles in `NBIN` bins, are:
(i) For each bin, calculate the signal correlation derivative matrices $\partial S/\partial C_b$; complexity is O(`NMPL` $\times$ `NPIX`^{2}).
(ii) Form and then invert the data correlation matrix $D = \sum_b C_b\, \partial S/\partial C_b + N$; complexity is O(`NPIX`^{3}).
(iii) For each bin, calculate $W_b = D^{-1}\partial S/\partial C_b$; complexity is O(`NBIN` $\times$ `NPIX`^{3}).
(iv) For each bin, calculate $\partial \mathcal{L}/\partial C_b$; complexity is O(`NBIN` $\times$ `NPIX`^{2}).
(v) For each bin-bin pair, calculate $\partial^2 \mathcal{L}/\partial C_b \partial C_{b'}$; complexity is O(`NBIN`^{2} $\times$ `NPIX`^{2}).
(vi) Invert the bin correlation matrix $\partial^2 \mathcal{L}/\partial C_b \partial C_{b'}$ and calculate the spectral correction δC_b; O(`NBIN`^{3}).

Note that phases *(iv)* to *(vi)* are computationally subdominant as `NBIN` << `NMPL` << `NPIX`.

2.2 MADbench2

The most computationally challenging element of the MADCAP package, used to derive spectra from sky maps, has also been recast as a benchmarking tool – MADbench2. 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 corresponding to the six phases of MADCAP:

- *Step 1:* Build a sequence of Legendre polynomial based CMB signal pixel-pixel correlation component matrices (one matrix for each bin), writing each to disk (loop over {calculate, write}) - this corresponds to MADCAP phase *(i)*;
- *Step 2:* Form and invert the full CMB signal+noise correlation matrix (calculate/communicate) - *(ii)*;
- *Step 3:* 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}) - *(iii)*;
- *Step 4:* In turn, read each pair of these result matrices from disk and calculate the trace of their product (loop over {read, calculate/communicate}) - *(iv)* to *(vi)*.

Due to the large memory requirements of the computational domain in real calculations, the matrices do not all fit in memory simultaneously, thus an out-of-core algorithm is used. Each processor requires enough memory to store just five matrices at any one time - the three matrices of the computation mentioned in Section 2.1 and two additional matrices that may act as targets for I/O during the computation. MADbench2 writes the individual component matrices to disk when they are first calculated, and rereads them as they are needed. Since the heart of the calculation is dense linear algebra, all the matrices are stored in the

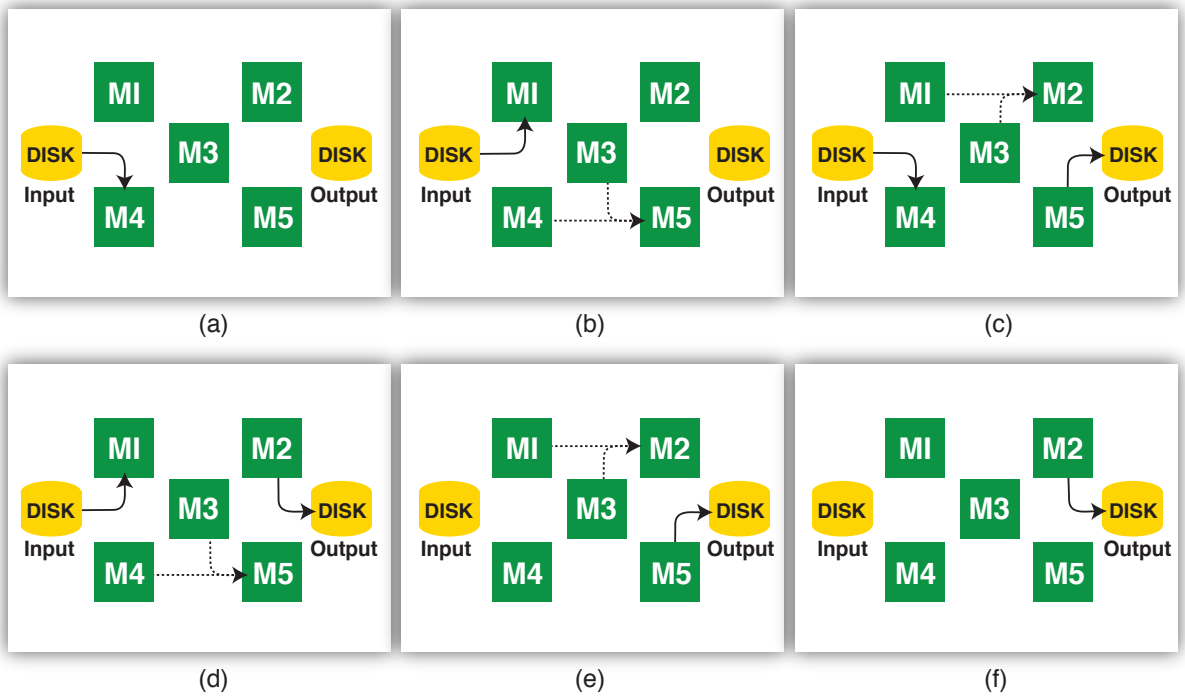


Figure 2: Step 3 visualization, showing computational data flow between the five matrices (dotted arrows) and I/O transfers (solid arrows). (a) Read a CMB signal correlation component matrix (b) Multiply by the inverse CMB matrix, while reading the next CMB matrix (c) Write result, while multiplying the next and reading the one after (d) Continue alternating matrices (e) Multiply the last while still writing (f) Write the last matrix.

ScaLAPACK [22] block-cyclic distribution, and each processor simultaneously writes or reads its share of each matrix in a single transfer.

Figure 2 presents a visualization of the data flow for *Step 3*. The process begins by reading a CMB signal correlation component matrix from disk in Figure 2(a). Next that matrix is multiplied by the inverse CMB matrix, while the next CMB matrix is read from disk in Figure 2(b). The result of that computation is written out to disk, while the next matrix is read from disk and the previously read matrix is multiplied in Figure 2(c). This process continues while alternating matrices (to save space) until the last matrix is read shown in Figure 2(d). The computation and matrix writing continues in Figure 2(e). Finally, the last matrix is written out in Figure 2(f).

Note that the data movement in *Step 3* is the most complicated. *Step 1* resembles *Step 3* but without matrix read operations, and *Step 4* conversely has no matrix write operations. *Step 2* does not involve I/O but may suffer from inter-process communication. This provides MADbench2 with an additional dimension of benchmarking for HPC systems on a real-world problem, previously explored in [3, 6]. In an environment that permits asynchronous I/O (see Section 5) the computation and I/O can proceed simultaneously. When only synchronous I/O is available, the data flow of Figures 2(c-d) take place in this order: write, compute, read. If asynchronous I/O is available, and for sufficiently large computations, a significant portion of the I/O can be hidden behind the computation.

In an *I/O experiment* no actual ScaLAPACK functions are called. Instead, a *busy-work* activity simulates the work. Furthermore, the *busy-work* computation is tunable, using $O(\texttt{NPIX}^{2\alpha})$ operations, α is the tuning parameter. An *I/O experiment* allows MADbench2 to investigate I/O performance on experimental systems where the usual linear algebra libraries are not yet installed. Tuning α to a very small value (less than 0.1) allows MADbench2 to run large I/O performance tests without having to wait for linear-algebra calculations. Tuning α to higher values (in the range 1 to 2) allows one to measure at what value of α the calculation time

matches the I/O time for a given system. MADbench2 measures the time spent in calculation/communication and I/O by each processor for each *Step*, and reports their average and extrema. Section 5 explores the effect of varying α on the ability of a system to hide the I/O.

2.3 MADbench2 Parameters

MADbench2 accepts environment variables that define the overall I/O approach as well as command-line arguments that set the scale, gang-parallelism, and system-specific tuning of any particular run:

- `IOMETHOD` - either POSIX or MPI-IO data transfers
- `IOMODE` - either synchronous (SYNC) or asynchronous
- `FILETYPE` - either unique or shared files, i.e. one file per processor versus one file for all processors
- `BWEXP` - the busy-work exponent α
- `NPIX` - the number of pixels
- `NBIN` - the number of bins
- `NOGANG` - the number of gangs, set to 1 throughout here
- `SBLOCKSIZE` - the ScaLAPACK block size, irrelevant here since we run in I/O-mode since we only run *I/O experiments*
- `FBLOCKSIZE` - the file blocksize, with all I/O calls starting an integer number of these blocks into a file (= 1048576)
- `RMOD/WMOD` - limiting the number of processes that read/write simultaneously. Total processors divided by RMOD (WMOD), determines the number of simultaneous readers (writers).[1]

The `NPIX` and `NBIN` set the problem size, resulting in a calculation with `NBIN` component matrices, each of $\texttt{NPIX}^2$ doubles. The last three arguments allow for some system-specific tuning. Each of the I/O calls is $O(8\,\texttt{NPIX}^2/\#\text{CPU})$ bytes, and each is fixed to start an integer multiple of `FBLOCKSIZE` bytes into the file. In the test traced in Figure 3 each matrix was about 300`MB`.

Note that although MADbench2 is originally derived from a specific cosmology application, its four patterns — looping over work/write (*Step 1*), communication (*Step 2*), 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 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 and 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 challenge 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.

To gain further insight into MADbench2's data flow, Figure 3 depicts the trace of a MADbench2 run on 16 processing elements (PEs) of the "Franklin" Cray XT4 at NERSC (detailed in Section 3.3). This test example ran synchronously and is an "I/O experiment" (no *Step 2*), using POSIX data transfers, one unique

[1] For all the systems studied here, optimal performance was seen for $\texttt{RMOD} = \texttt{WMOD} = 1$, where every process writes or reads simultaneously.

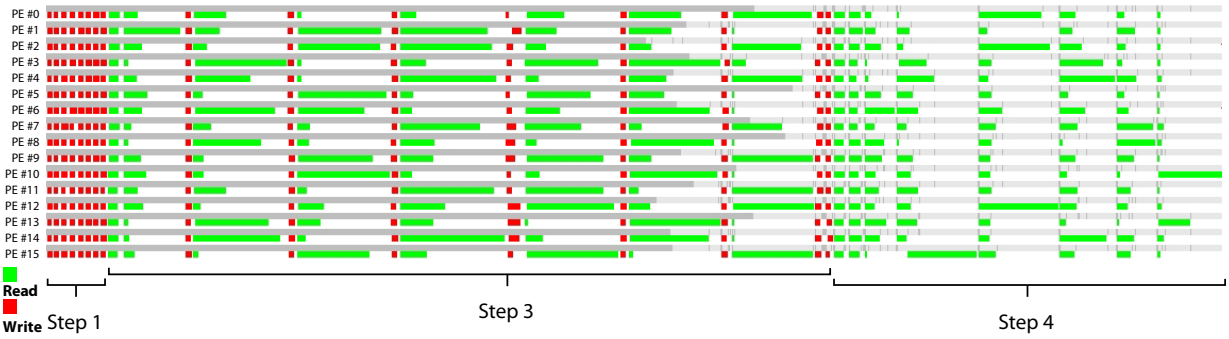


Figure 3: A 16 CPU MADbench2 run with 8 bins

file per processor, small *alpha* = 0.010, `NPIX` = 25,000, `NBIN` = 8, `NOGANG` = 1, `FBLOCKSIZE` = 1048576, and `RMOD/WMOD` = 1. Recall that to allow asyncronicity there are two initial reads before the first write, and two consecutive finalizing writes. The trace in Figure 3 also shows that the writes are short and consistent in duration compared to the reads. The extreme variability and long duration of the reads is an ongoing subject of investigation. Note that the first read for each file is relatively short and consistent. Subsequent reads, each of which requires a seek far into the file, take longer and are more variable. This would seem to indicate that the underlying file system (Lustre) is suffering from occasional performance degradation on reads from very large files. A detailed analysis of the low-level system behavior and variability will be the focus of future work.

Though all of the tests reported here are *I/O experiments*, which explore I/O performance and its relation to compute performance, one should note that MADbench2 is also useful for exploring communication performance. 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, 6].

2.4 Related Work

There are a number of synthetic benchmarks investigated for studying I/O performance on HPC systems. Unfortunately, most of the existing benchmarks are not reflective of the current I/O practice of HPC applications, either because the access pattern does not correspond to that of the HPC applications, because they only exercise POSIX APIs (e.g.. no MPI-IO, HDF5, NetCDF), or because they measure only serial performance.

IOZone [15] is a popular file system benchmark used to measure the performance of a variety of file operations. It is designed to understand the file system's I/O characteristics from the standpoint of manufacturer's I/O performance metrics, but it is difficult to directly relate its results back with applications. Likewise, HPIO [13] is a synthetic benchmark designed to measure noncontiguous I/O performance, but does not target any specific application. FileBench [18] is a more flexible I/O system benchmark which provides a workload model language to enable the creation of diverse workload benchmarks, however, it lacks the parallel programming interfaces needed by scientific applications, such as MPI-IO, HDF5, or Parallel-NetCDF. Also many access patterns are targeted at transaction processing and online database applications, which are very different from the requirements of HPC applications.

The Effective I/O Benchmark [20] reports an I/O performance number by running a set of predefined configurations to derive a single performance number as its performance metric. However, the predefined composite metric does not cover the full parameter space of I/O patterns demonstrated by the HPC applications we investigated. MPI-Tile-IO [27] tests access to a two dimensional dense dataset. NAS-IO [28]

includes the BT I/O benchmark to test the I/O performance of the Block Tridiagonal problem of the NPB. However, these benchmarks focus on a relatively narrow set of target applications. They also focus exclusively on the MPI-IO interface. The SPIObench [26] and PIORAW [19] are parallel benchmarks that have been used successfully to assist in HPC system evaluations. Both of these benchmarks read and write to separate files in parallel fashion, but do not offer a way to assess performance for concurrent access to a single file. SPIObench is used in the Department of Defense HPC Modernization Program procurements and includes a useful feature in that it allocates extra memory on each node so as to avoid artificially high read/write performance from block buffer caching.

The IOR [14] 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. Preliminary work has shown that IOR can be a reasonable proxy for synchronous MADbench behavior [25]. However, IOR measures the read and write performance separately, thus restricting its ability to model MADbench2's concurrent read-write operations, particularly in asynchronous mode.

The common theme with the above synthetic benchmarks is that they are able to measure components of the I/O system and mimic some subset of applications, but have limited ability to model the range of I/O behavior of demanding HPC applications.

The FLASH-IO [9] benchmark, studied extensively in a paper on paralleNetCDF [7], 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. A recent comparison of I/O benchmarks can be found in a study by Saini, et al. [21].

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 eight leading HPC systems. These platforms include seven architectural paradigms — Cray XT3, Cray XT4, IA64 Quadrics cluster, Power5 SP, Opteron Infiniband cluster, BlueGene/P (BG/P), and SGI Altix — and four state-of-the-art parallel file systems — Lustre [5], GPFS [24], PVFS2 [16], and CXFS [8]. Additionally, we examine I/O scalability to 1024 processors on the recently-released Franklin XT4 and BlueGene/P systems.

In order to minimize the communication overhead of *pdgemm* calls, at each concurrency the tests reported here set `NPIX` to fill the available memory. Since file systems generally scavenge otherwise unused memory to buffer I/O, many benchmarks will show data-rates that exceed the system's theoretical peak performance until memory is filled. Such high data rates lead to unrealistic expectations for most real applications, so this choice of `NPIX` also minimizes the opportunity for system level buffering to hide I/O costs.

CPU	`NPIX`	`NBIN`	Mem (GB)	Disk (GB)
16	25,000	8	23	37
64	50,000	8	93	149
256	100,000	8	373	596
1024	200,000	8	1490	2381

Table 1: Configurations with associated memory and disk requirements for I/O tests with MADbench2.

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 (312 MB/CPU/call) across all concurrencies.

As described in Section 2.1, `NBINS` is dictated by the quality and coverage of the data in an actual MADCAP calculation. Even though large-scale MADCAP calculation may require large numbers of bins, our experimental tested uses `NBINS` = 8 in all cases. This choice balances the need to keep tests relatively short with the desire to show the recurring pattern in the calculation. For synchronous experiments, fewer `NBINS` would not reveal useful results for the MADCAP calculation, while more `NBINS` would waste compute resources carrying out additional iterations identical to those in Figure 3. However, `NBINS` does correspond the potential of hiding asynchronous I/O, as detailed in Section 5.

The systems considered here typically have 2 GB/CPU, of which our chosen problem sizes filled 73%. The BG/P nodes have 2GB of memory shared by four CPUs. Since the tests reported here emphasize I/O, only one CPU per node was used on BG/P; 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 File Systems

In this section, we briefly outline the I/O configuration of each of the evaluated supercomputing systems. Table 2 summarizes the features of the file system under review. Figure 4 diagrams the difference between a file system architecture in which *Storage Servers* mediate the access to the underlying disk versus an architecture in which the underlying block device is mounted directly on the file system clients.

The demands of coordinating parallel access to file systems 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 file systems 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 file system 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 processes 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.

In Lustre the Object Storage Target (OST) is a process running on a storage server that mediates access to the underlying disk device. Lustre refers to a bulk I/O server as an Object Storage Server (OSS - **(a)** in Figure 4), where each OSS may have several OSTs. GPFS refers to a server as a Network Shared Disk (NSD). PVFS refers to a server as simply a "data server". GPFS can also use a Virtual Shared Disk (VSD) from a server in a fashion resembling the CXFS file system architecture (**(b)** in Figure 4), so that each client communicates directly with a block device (VSD).

The front-end of the I/O servers (both bulk I/O and metadata) are usually connected to the compute nodes via the same interconnect fabric that is used for message passing between the compute nodes. The BG/P systems are an exception to this, as they use a dedicated Tree/Collective network to connect the compute nodes to I/O nodes within their partition. The BG/P I/O nodes (IONs) act as the storage clients and connect to the I/O servers via 10GigE links. 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.

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
Franklin	Lustre	Opteron	SeaStar2	6.4	1.2	1:112	10.2	350
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
Columbia	CXFS	Itanium2	FC4	1.6	—*	—*	1.6†	600
Intrepid	GPFS	BG/P	10GigE	0.35	0.3	1:64	8	233
Surveyor	PVFS2	BG/P	10GigE	0.2	0.2	1:64	1.0	88

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. *Inapplicable to Columbia's SAN implementation. †Although 3.0+ GB/s are available across the system, only 1.6 is available to each 512-way SMP.

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 file system. Each XT3 node contains a dual-core 2.6 GHz AMD Opteron processor, which is 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 to 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 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[2].

3.3 Franklin: Lustre on Cray XT4

Franklin, a 9,660 compute node Cray XT4 supercomputer that is located at Lawrence Berkeley National Laboratory (LBNL), also employs Lustre as the global file system. Each XT4 node contains a dual-core 2.6 GHz AMD Opteron processor, tightly integrated to the XT4 interconnect via a Cray SeaStar-2 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 to its six nearest neighbors. The XT4 uses Compute Node Linux (CNL) on compute processing elements (PEs) and SuSE SLES 9.0 on service PEs.

Franklin has 20 Object Storage Servers (OSS's), and one MDS, connected via the custom SeaStar-2 network in a toroidal configuration to the compute nodes. The OSSs implement a total of 80 OSTs that use Data Direct Networks (DDN) 9500 RAID couplets as backend block devices for the OSTs. DDN9500 couplets have a published peak rate of 3 GB/s, though depending on the load characteristics the delivered rate may be closer to 1.5 GB/s to 2.0 GB/s. The scratch file system targetted in this testing uses five copulets with a measured aggregate I/O rate of about 10 GB/s.

3.4 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 file system. 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.

[2]Personal communication Jaguar system administrator.

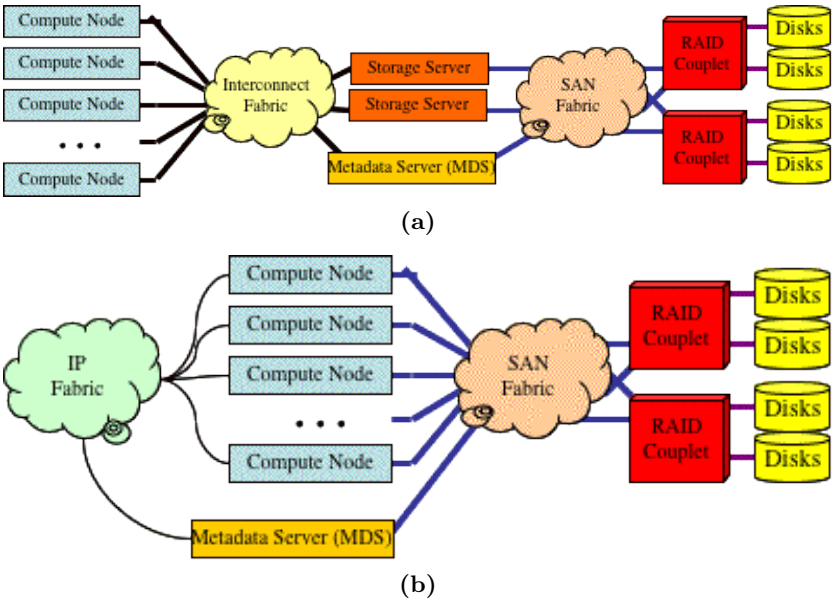


Figure 4: The general architecture of studied cluster file systems, including Lustre, PVFS2, and GPFS, follow the abstract pattern shown in (a); the CXFS configuration is more similar to a SAN, shown in (b).

Thunder's 16 I/O OSS servers (or GateWay 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.

3.5 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 file system. 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. The NERSC PIORAW benchmark [19] measures the aggregate disk throughput to be $4.4GB/s$ for reads and $4.1GB/s$ for writes.

3.6 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 file system that employs 16 (out of a total of 20) I/O servers 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 file system examined in our study) have 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 by the IP over IB implementation, and sustains only ~200-270 MB/s per DDN.

Thus the aggregate peak is expected to be closer to $4.5GB/s$, and has been measured (by PIORAW) at $4.2GB/s$ for reads and writes.

3.7 Columbia: CXFS on SGI Altix3700

Columbia is a 10,240-processor supercomputer that is located at NASA Ames Research Center and employs 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" that 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.

3.8 Intrepid: GPFS on BG/P

Argonne National Labs (ANL) has recently deployed two BG/P machines. Intrepid is the larger, production machine with 8192 nodes. Each BG/P node has 4 PowerPC 450 CPUs (0.85 GHz) and 2GB of memory. As mentioned in Section 3.1, the I/O architecture of BG/P systems is designed to scale with the size of the system. On Intrepid, 64 BG/P compute nodes (CNs) access the external I/O resources through one BG/P I/O node (ION) which acts as a client for the parallel file system. Other BG/P deployments may have CN to ION rations of 8, 16 or 32. The resource scheduler allocates such blocks of 64 CNs as a unit, so larger jobs will see a higher bandwidth to the external I/O resources. Each ION has a 10Gb Ethernet interface connecting to the storage server network. The full 8192 nodes of Intrepid have 128 IONs connected to the GPFS storage network.

The GPFS file system on Intrepid is served by 16 IBM x3655 file servers (12GB RAM, 2 dual core 2.6GHz CPUs), with each server fronting 8 LUNs of 4TB each. The peak data rate through a single ION is about 350MB/s with a sustained value closer to 300MB/s. The aggregate for the file system is about 8000MB/s, so the external I/O resource is expected to be saturated at about 25 IONs or 1600 nodes. Plans are currently in the works to extend the external I/O resource, thereby greatly expanding the aggregate I/O capability of the system as a whole.

3.9 Surveyor: PVFS2 on BG/P

Surveyor is ANL's smaller, development cluster with 1024 nodes. Surveyor uses the PVFS2 file system for its scratch file system and employs 16 IONs. Survey also allows allocations as small as 16 nodes. The PFVS2 scratch file system on Surveyor comprises 4 LUNs of 24TB each served by a single IBM x3655 server. It can provide a peak aggregate I/O bandwidth of about $1000MB/s$ for writes and $600MB/s$ for reads.

4 Synchronous I/O Behavior

This section presents performance on our test suite using the synchronous I/O configuration, where computation and communication are 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 file systems 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 file systems 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.

Figure 5 presents a breakdown of each individual system's performance, while Figure 6 shows a summary of synchronous MADbench2 results on the eight evaluated architectural platforms using the default environment parameters. Detailed performance analysis and optimized execution results are presented in the subsequent subsections.

4.1 Jaguar Performance

Figures 5(a-b) present Jaguar's synchronous performance. For unique file reading and writing the Cray XT systems Jaguar and Franklin show the best performance. In particular, Jaguar shows a significantly higher peak aggregate I/O rate than the other architectures for both reads and writes. As with most cluster file systems, 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 5(a), increasing concurrency to 64 nodes causes the Jaguar I/O system to approach the 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 use 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 5(a). Thus, its default shared performance is relatively poor compared with Bassi, and Jacquard, as shown in Figure 6(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 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 shown in Figure 5(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 Franklin Performance

Figure 5(d-e) presents the synchronous performance on the Franklin system. Like Jaguar, Franklin does not reach peak I/O performance until the number of client nodes is able to aggregate sufficient interconnect bandwidth to saturate the back-end storage subsystem. However, Franklin has fewer object storage servers than Jaguar (20 on Franklin compared with 48 on Jaguar), allowing it to reach saturation at a lower concurrency than Jaguar. Consequently Jaguar continues to see performance improvements between 64 and 256 processors (Figure 5(a)), while Franklin reaches its peak I/O rate at 64 and performance declines

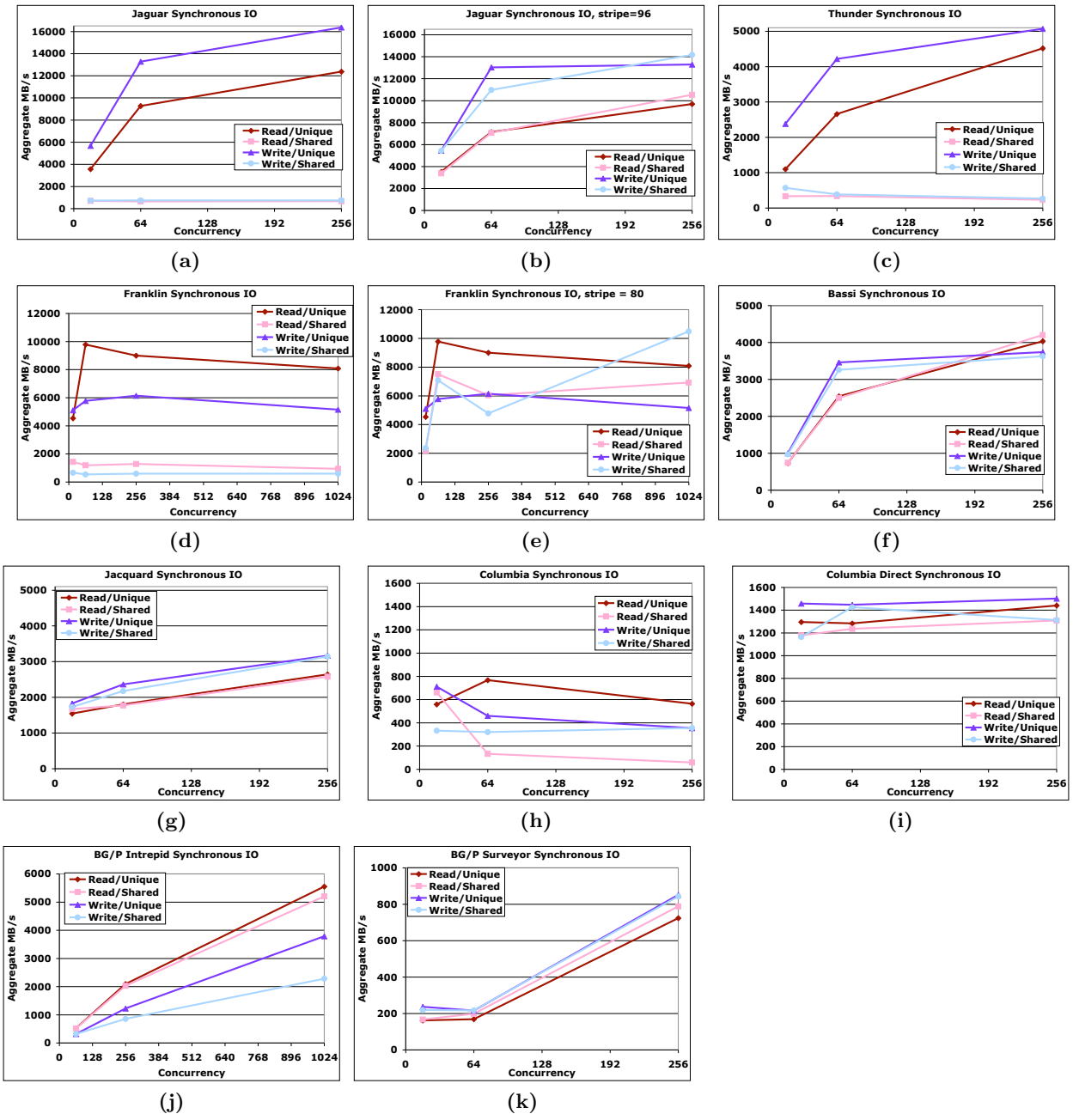


Figure 5: MADbench2 performance on each individual platform showing : (a) Jaguar default environment, (b) Jaguar optimized striping configuration, (c) Thunder default environment, (d) Franklin default environment, (e) Franklin optimized striping configuration, (f) Bassi default environment, (g) Jacquard default environment, (h) Columbia default environment, (i) Columbia using directIO interface, (j) Intrepid default environment, (k) Surveyor default environment. Note that the aggregate I/O (y-axis) is scaled according to each architecture's performance, and the maximum concurrency (x-axis) is 256 processor for some architectures and 1024 for others.

modestly for 256 processors as resource contention for the OSS increases.

Examining the 1024-way Franklin experiments, Figure 5(d) shows the I/O performance continues to see a modest degradation. This is likely due to increasingly uncoordinated presentation of I/O requests to the storage servers (as the number of clients is increased), as well as the additional stress on the metadata server that effectively must serialize the file creation requests for the one-file-per processor (unique-file) case.

Like Jaguar, the shared file I/O performance is uniformly flat using the default environment settings. Whereas the default configuration spreads the file across 4 OST's, the `lstripe` command enables the file to aggregate the performance of all 80 OSTs as shown in Figure 5(e). Overall Franklin's global file system can effectively maintain scalable performance to high concurrencies for the shared files, while the I/O rate of the unique-file experiments modestly decreases at high concurrencies. The anomalous performance dip at 256 processors is currently under investigation.

Comparing aggregate unique-file performance in Figure 6(a-b), it can be seen that Franklin achieves comparable (or higher) performance for smaller numbers of processors. However, at high concurrencies Jaguar sustains a significantly higher I/O rate, due to additional object storage servers and a 2x advantage in peak disk bandwidth. Thus for unique-file accesses, Franklin is the second fastest system in our study. Looking at the shared file experiments in Figure 6(c-d), it can be seen that — much like Jaguar — Franklin's default performance drops dramatically without explicit striping. As a result it sustains a lower aggregate I/O rate than the GPFS systems in our study (Bassi, Jacquard, Intrepid).

4.3 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 6 Thunder achieves the third highest aggregate performance after Jaguar and Franklin for unique file accesses (at higher concurrencies). The relative drop off compared to Jaguar is not surprising since Thunder's peak I/O performance is only one-fifth that of Jaguar's. Figure 5(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. 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 difference between Lustre performance on Jaguar and Thunder is largely due to the much older software and hardware implementation deployed on Thunder.

4.4 Bassi Performance

As seen in Figure 5(f), and unlike the Lustre-based systems, Bassi's GPFS file system 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 6), 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 the I/O servers. With 6 I/O servers, 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.

4.5 Jacquard Performance

The Jacquard platform provides an opportunity to compare GPFS performance on a system with both architecture and OS different from the PowerPC and AIX combination on Bassi, which was the original

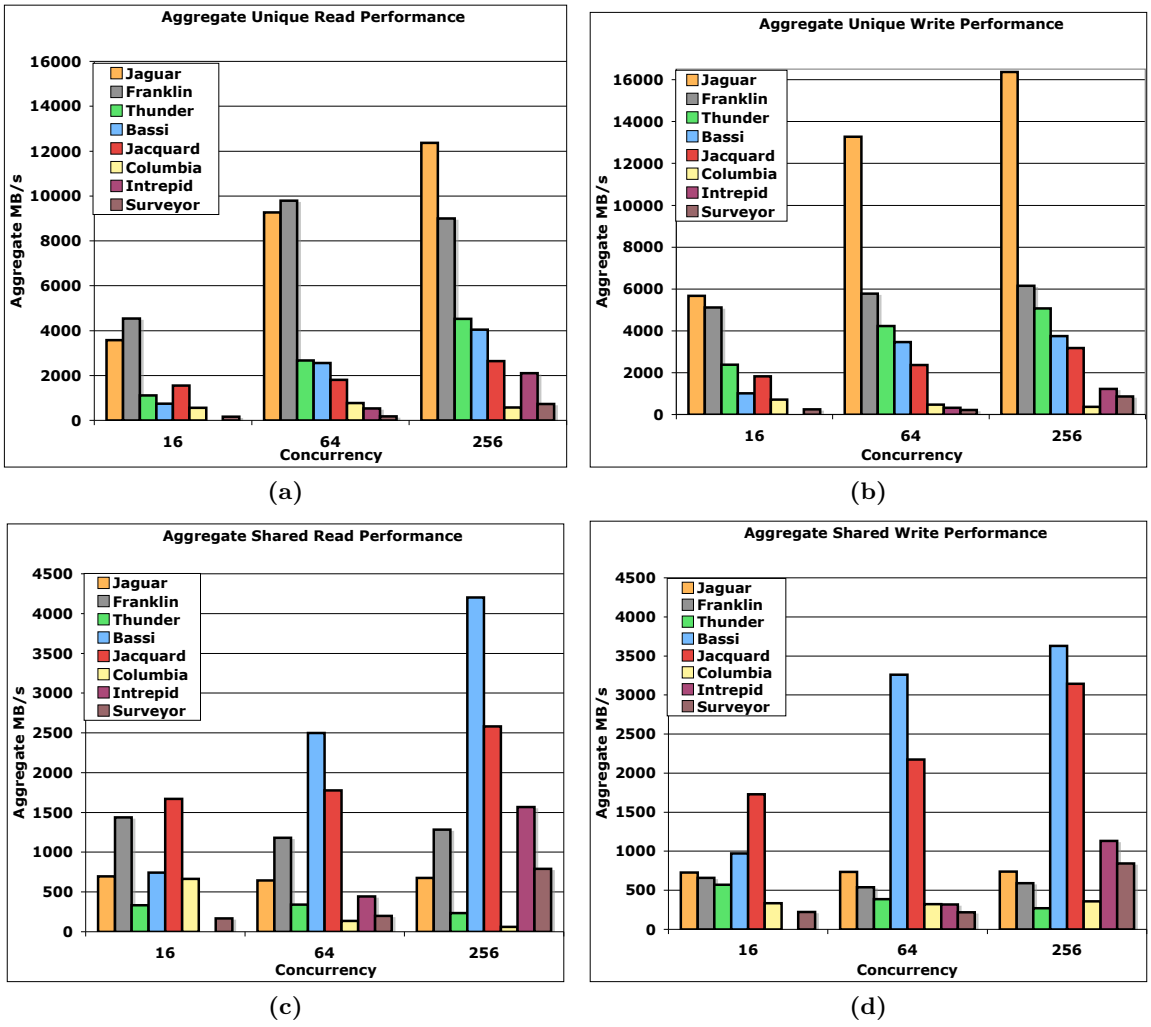


Figure 6: Synchronous MADbench2 aggregate throughput results for fixed concurrencies using the default environment parameters, showing unique file (a) read and (b) write performance, and shared file (c) read and (d) write performance.

development target for GPFS. Figure 5(g) 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 6, Bassi generally outperforms Jacquard due to its superior node to I/O bandwidth (see Table 2).

4.6 Columbia performance

Figure 5(h) shows the baseline performance of the Columbia CXFS file system. 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 physical 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 6.

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 5(i) 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.

4.7 Intrepid performance

Figure 5(j) presents Intrepid GPFS performance results, showing significant scaling behavior from 64 to 1024 nodes. Recall that 64 nodes of a BG/P share a single ION to the external parallel file system (64-way experiments are the smallest available allocation unit on Intrepid). Observe that unlike most other systems, read performance is above write performance at all concurrencies. As seen with the other GPFS systems in our study, Intrepid's file system shows shared file performance for reads that is close to unique file performance. As a result, the comparative data in Figure 6 shows that Intrepid's aggregate shared performance (at 256 nodes) exceeds all the architectures in our study, except for the Bassi and Jacquard GPFS systems.

Finally, we highlight that at 1024 nodes Intrepid's performance nears the 8 GB/s peak rate available from the external GPFS file system (Figure 5(j)). This bodes well for the planned expansion of the GPFS file system, which targets an aggregate capacity of 88 GB/s to balance the 8192 nodes of the Intrepid system.

4.8 Surveyor performance

Surveyor's performance behavior is shown in Figure 5(k). Since each group of 64 BG/P CNs share a single ION, the Surveyor PVFS2 system shows the same reading and writing performance for 16 and 64 nodes. Unlike Columbia, however, each group of 64 CNs results in additional IONs and consequently increased bandwidth. With four times the number of IONs, the 256-way experiment sustains four times the bandwidth. Although the PVFS2 file system on Surveyor is modest, it does not appear to be saturated at 256-way concurrency. In fact, shared write performance is comparable to the much larger Intrepid system for the 256-way experiments, as seen in Figure 6 — outperforming Franklin, Thunder, and Columbia. However, Surveyor's unique performance is the second lowest in our study, only exceeding that of the Columbia system

5 Asynchronous performance

Almost all of the systems investigated in our study show sub-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.

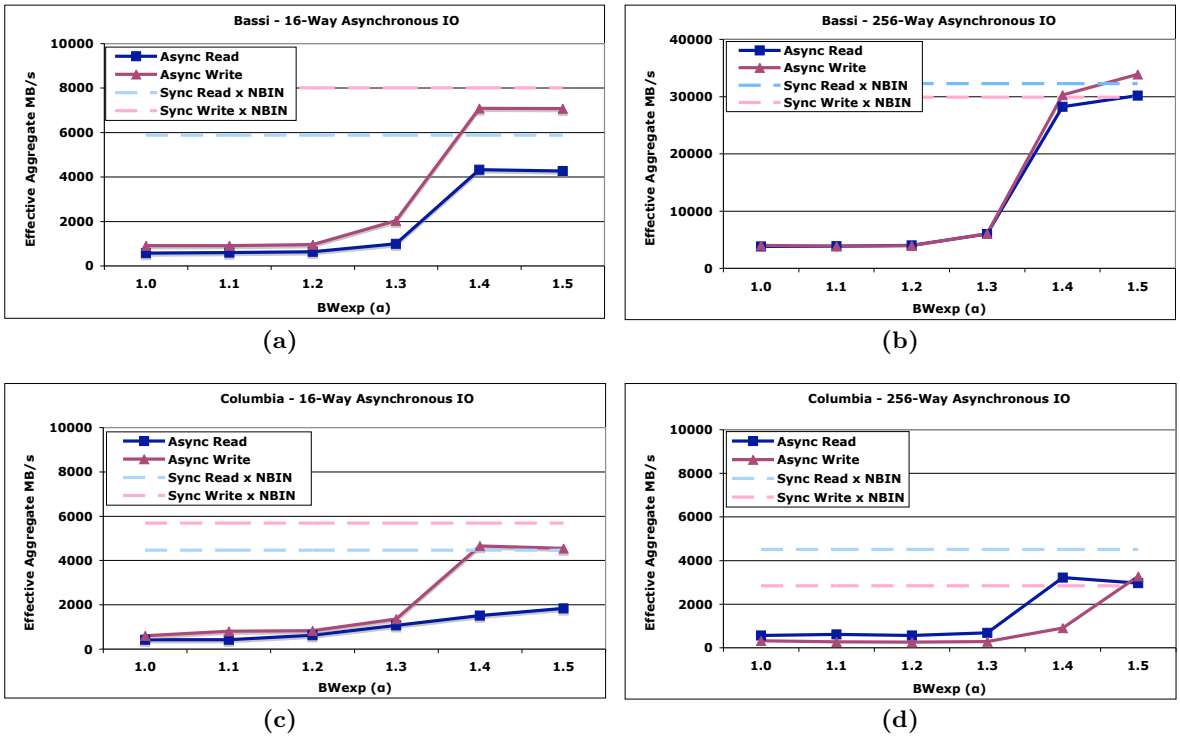


Figure 7: 16-way and 256-way *effective aggregate asynchronous rate* as a function of the busy-work exponent, for the two evaluated platforms supporting this functionality (a-b) Bassi and (c-d) Columbia. The busy-work exponent ($\boldsymbol{\alpha}$) of 1.0 and 1.5 correspond to BLAS2 (matrix-vector) and BLAS3 (matrix-matrix) computational intensities (respectively).

One possible way to ameliorate this potential bottleneck is to hide the I/O cost behind simultaneous calculations by using asynchronous I/O. This technique is applicable to any application where I/O transfers can be overlapped with computational activity. The design of MADbench2 includes an option to employ asynchronous I/O, using portable MPI-2 [10] 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). POSIX asynchronous I/O (AIO) is not examined in this work as MPI-I/O is a more portable interface, and both MPI and POSIX would likely use the same underlying system's asynchronous facilities. Future work will examine AIO characteristics on platforms that support asynchronous I/O behavior.

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 (matrices of $\texttt{NPIX}^2$ doubles in MADbench2). 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 systems 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 overhead. As a measure of the benefit that backgrounded I/O represents, we introduce the *effective aggregate asynchronous rate* (EAAR) for a MADbench2 experiments, as the amount of data moved divided by the foreground I/O time. Figure 7 presents the EAAR as function of α. The lines

labeled `Sync Read x NBIN` and `Sync Write x NBIN` represent the potential target speedup by a factor of `NBIN` compared with the values from Figure 5.

The results in Figure 7 demonstrate the potential gain of an asynchronous I/O implementation. With high α, Bassi's EAAR (Figure 7(a-b)) exceeds the synchronous I/O rate of other systems investigated in our study (for both read and write, at all concurrencies), even doubling Jaguar's I/O at 256 processors. Similarly, Columbia's EAAR (Figure 7(c-d)) exceeds its synchronous rate by a factor of 8 for reading and writing across all concurrencies; however, Columbia's failure to scale I/O indicates that the file system 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 examined 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 volumes. Our study represents one of the most extensive I/O analyses of modern parallel file systems, exploring a broad range of system architectures and configurations. These performance results are a usful measure of file system value along with such factors as the costs of hardware, licensing, and maintenance.

Our work introduces 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 an *I/O experiment*) 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. Additionally, we have made the code publicly available [17] to facilitate other institution's investigations of global file system behavior.

Results show that, although concurrent accesses to shared files in the past required MPI-IO for correctness on clusters, the modern file systems 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 file systems — you can (and should) expect file system 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 file systems, 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 ANL BG/P system did not reach saturation even at the maximum tested concurrency of 1024 processors. Additionally, results on Franklin showed saturation at 64 processors, but relatively little degradation up to 1024 processors, despite the increased stress on the storage and metadata servers.

Finally, we found that MADbench2 was able to derive enormous file system performance benefits from MPI-2's asynchronous MPI-IO. Unfortunately, only two of our nine 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 computation 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 concern 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 file system performance to quantify I/O variability under differing conditions.

Acknowledgments

The authors would like to thank the following individuals for their kind assistance in helping us understand and optimize evaluated file system performance: Tina Butler and Jay Srinivasan of LBNL; Richard Hedges of LLNL; Nick Wright of SDSC; Susan Coughlan, Robert Latham, Rob Ross, Andrew Cherry, and Vitali Morozov 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.

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] J. Borrill, L. Oliker, J. Shalf, and H. Shan. Investigation Of Leading HPC I/O Performance Using A Scientific-Application Derived Benchmark. In *Proc. SC07: High performance computing, networking, and storage conference*, Reno, NV, 2007.

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

[6] 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.

[7] 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.

[8] 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.

[9] Flash io benchmark. `http://www-unix.mcs.anl.gov/pio-benchmark/`.

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

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

[12] 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.

[13] Hpio the high-performance I/O benchmark. `http://cholera.ece.northwestern.edu/~aching/research_webpage/hpio.html`.

[14] The ASCI I/O stress benchmark. `http://sourceforge.net/projects/ior-sio/`.

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

[16] 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.

[17] MADbench2: A Scientific-Application Derived I/O Benchmark. `http://outreach.scidac.gov/projects/madbench/`.

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

[19] Performance Testing with the PIORAW Benchmark. `http://www.nersc.gov/nusers/systems/bassi/perf.php`.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] H. Shan, K. Anypas, and J. Shalf. Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *Proc. SC2008: High performance computing, networking, and storage conference*, Austin, TX, Nov 15-21, 2008, to appear.

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

[27] MPI-Tile-I/O. `http://www-unix.mcs.anl.gov/pio-benchmark/`.

[28] P. Wong and R. F. Wijngaart. NAS parallel benchmarks I/O version 2.4. In *Technical Report NAS-03-002, Computer Sciences Corporation*, NASA Arms Research Center, Moffett Field, CA 94035-1000, Jan, 2003.