Performance of the Vesta Parallel File System

Dror G. Feitelson Peter F. Corbett Jean-Pierre Prost
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

Abstract

Vesta is an experimental parallel file system implemented on the IBM SP1. Its main features are support for parallel access from multiple application processes to a file, and the ability to partition and re-partition the file data among these processes. This paper reports on a set of experiments designed to evaluate Vesta's performance. This includes basic single-node performance, and performance using parallel access with different file partitioning schemes. Results are that bandwidth scales with the number of I/O nodes accessed, and that orthogonal partitioning schemes achieve essentially the same performance. In many cases performance equals the disk hardware limit. This is often attributed to prefetching and write-behind in the I/O nodes.

1 Introduction

Massively parallel processors (MPPs) provide applications with high computing power by harnessing multiple microprocessors to work on the same job. To be effective, this large computing power must be balanced with adequate I/O capabilities [12]. This can be done by harnessing multiple disks to provide I/O services in parallel. Indeed, most MPP vendors include parallel I/O in their architectures. This is done in the form of dedicated I/O nodes, each with several disks, which are distinct from the compute nodes used to run applications. Examples include the Connection Machine CM-5 from Thinking Machines [20], The Intel Paragon, and the Meiko Computing Surface CS-2.

The system software on MPPs typically exploits the parallel I/O by striping file data across the I/O nodes, as first suggested in the Bridge file system [11]. Such striping is transparent as far as applications are concerned. It achieves the goal of parallel access to disks, but hides all the details of the striping from the application. Examples include Intel's CFS on the iPSC [26] and PFS on the Paragon, the sfs file system on the CM-5 [22], the nCUBE system software [8], and the Meiko parallel filesystem.

The Vesta parallel file system is based on the premise that the parallel I/O capabilities should be exposed at the file system interface, rather than being hidden. Specifically, Vesta provides control over the data layout on the I/O nodes, so applications can tailor the layout to match their access patterns [4, 6]. Vesta files have a 2-D structure to facilitate this level of control: one dimension represents *cells*, which are an abstraction of disks (these can be thought of as virtual disks, just like processes are sometimes referred to as virtual processors). The second dimension is data within cells. Vesta supports the common rectilinear decompositions of this 2-D structure, so application processes can access the data in a single cell, in a band across all the cells, or in blocks that contain some data from some cells. The system is fully implemented as of September 1993. It is now in initial stages of use in prototype applications, e.g. for the analysis of data from high-energy physics experiments at Fermi National Lab.

This paper describes performance experiments designed to test Vesta and the degree to which it utilizes the underlying hardware. The next section describes Vesta and the experimental setting. Section 3 analyzes the basic message passing characteristics of the system, for one compute node and one I/O node. Section 4 shows the effect of buffering on the I/O nodes. Section 5 deals with scalability of parallel access. Sections 6 and 7 present Vesta performance results relating to data partitioning, and Section 8 shows the performance benefits of asynchronous I/O. Finally, Section 9 offers performance results of a sorting application.

2 Background

2.1 Related Work

I/O subsystems have been neglected as a field of study for many years. However, with the rapid performance improvements in other components of computer systems, most notably processors, I/O is gaining recognition as the next major bottleneck — and consequently, a focus of research activity.

A major outcome of this new research is the development of RAID — redundant arrays of inexpensive (or independent) disks [15]. This technology uses parallelism within the I/O subsystem to compensate for the growing gap between the performance of processors and that of disks. It uses redundant data encoding schemes to compensate for the increased vulnerability to failures that comes with the parallelism [23, 14, 19]. It should be noted, however, that RAID does not necessarily imply a parallel interface. On the contrary, in most cases the parallelism is hidden in the implementation of the I/O subsystem, while using a traditional sequential interface. Thus the RAID provides the traditional abstraction of a single disk, just with a higher transfer rate and reliability.

In MPPs this is not enough. I/O can be done by many processes at the same time, accessing different or overlapping parts of the same files. A traditional interface would then become a bottleneck. The solution adopted by practically all vendors is a parallel I/O subsystem, often based on dedicated I/O nodes. These nodes have a number of disks attached to them (possibly a small RAID), and act as I/O servers. The compute nodes running application processes communicate with the I/O nodes via the same internal high-performance networks used for communication among the compute nodes. Examples of this architecture include the Connection Machine CM-5 [20], the Intel Paragon, and the Meiko CS-2.

A number of benchmarks for evaluating I/O performance have been designed over the years. These are surveyed and criticized by Chen and Patterson [3], who also suggest a new self-scaling benchmark. Unlike the previous ones, this new benchmark has the facilities to automatically change the workload used for the measurement, so as to stress the I/O system. As an example, they use the benchmark to compare the I/O performance of three workstations.

One of the configurable parameters in the self-scaling benchmark is the number of processes performing I/O, but this is only used to increase the activity so as to saturate the I/O subsystem. In order to evaluate a parallel I/O system, additional parameters need to be modified. For example, it is necessary to investigate different degrees of parallelism in the I/O system itself, and different patterns of interactions among the compute nodes and I/O nodes. As parallel I/O systems are relatively immature, there is no general agreement about exactly what should be measured. In addition, the measurements are often conducted within the confines of a specific installation, so they are limited to a single system configuration. Hence performance measurements for parallel I/O systems have been sketchy and hard to compare to each other.

Only the most basic results have been reported for sfs, the parallel file system on the Thinking Machines CM-5. These show how bandwidth scales as the number of disks used increases from 16 to 118, for transfer sizes of up to 230 MB [22]. Performance results reported for the nCUBE I/O system concentrate on how the performance degrades if the data is permuted as part of the transfer. This happens both when the system is used to implement parallel pipes (with no disk access) [9], and when it is used to store data on disk [10]. The

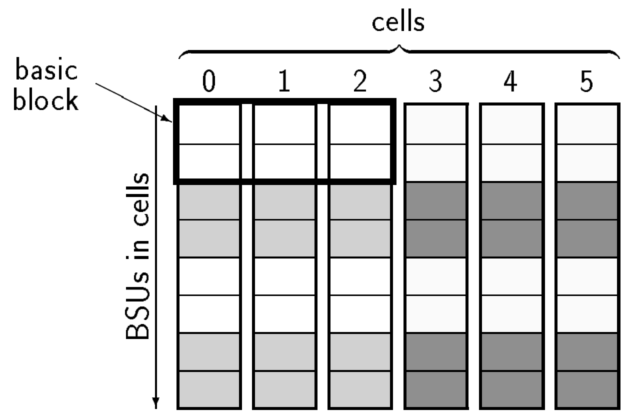


Figure 1: *Vesta partitioning example. A file with 6 cells is partitioned into four subfiles (different shades of gray), using a basic block of 2 BSUs from each of 3 consecutive cells.*

most widely studied system is Intel's CFS, which is available on the iPSC hypercubes and on the Touchstone Delta. The original description of CFS includes results relating to bandwidth dependence on the number of compute nodes and I/O nodes, with up to 16 and 8 of each type, respectively [26]. Similar measurements were conducted on the iPSC/860 at NASA Ames, which has 128 compute nodes and 10 I/O nodes. These measurements showed that when the ratio of compute nodes to I/O nodes becomes too large, thrashing might occur because the caches on the I/O nodes are too small to contain independent sets of blocks for all the compute nodes [25]. Another study, on a 32-node iPSC/2 with 4 I/O nodes, emphasized the variance between the service provided to the different compute nodes [13]. Finally, measurements of CFS on the Touchstone Delta (a 16×32 mesh with 32 I/O nodes and 64 disks, that served as a prototype for the Paragon) compared the different access modes provided by the system [1].

In this paper we investigate the performance of the Vesta parallel file system. Like previous work, we include measurements of bandwidth dependence on the number of I/O nodes and compute nodes. Additional measurements are biased by the special features of our system, specifically the mechanisms to partition files and interleave the accesses from different compute nodes. We attempt to provide a comprehensive characterization of all aspects of the system, and show how each element contributes to the overall performance.

2.2 Vesta Interface and Implementation

Vesta is implemented using a client-server structure. The client part is a library that is linked with application processes on the compute nodes. The server is a standalone executable intended to run on the I/O nodes [5].

The client part is responsible for access to the 2-D Vesta files. Recall that files are composed of one or more cells. The data in each cell is viewed as a sequence of *basic striping units*, or BSUs. When a file is opened, a set of 4 parameters is given [4, 6]. These parameters define the data decomposition, much like the way data decompositions for 2-D arrays are defined in HPF [21]. Specifically, two parameters define the number of cells and BSUs spanned by the basic block used in the decomposition, and the other two define how many such blocks there are in the horizontal and vertical dimensions (see Fig. 1). Thus the four parameters indicate how the file is decomposed into a set of disjoint *subfiles*. A fifth parameter specifies which of these subfiles is being opened. The open call returns a file descriptor that can be used to access that subfile.

Once a subfile is open, the client can access the data in it using Vesta read and write calls, which are similar to Unix read and write. The data in the subfile is logically treated as a single sequence, even though it may be physically distributed across multiple cells. For each access request, the client communicates with the I/O nodes that contain the relevant file cells to facilitate the data transfer.

The servers implement the cell abstraction and perform the actual disk I/O. The main issue in cells is maintaining the mapping from logical offsets in the cell to disk blocks. This is done using a tree-structured block list, which is somewhat reminiscent of a Unix inode [5]. The Vesta block size is 64 KB. Blocks that are accessed are kept in a 32 MB buffer cache on each I/O node (this is a system configuration parameter). An access counter is maintained for each block. This counter records the number of bytes that were read from or written to the block. When the counter indicates that the whole block was read, it is placed on a list of candidates for replacement. When the counter indicates that the whole block was written, it is written asynchronously to disk. This is known as the WriteFull policy [16]. Dirty blocks are also written to disk if a low-water-mark of free buffer slots is reached, or if load on the node is low, regardless of whether or not they are full.

The Vesta servers also implement a prefetching mechanism for sequentially accessed data. This is applicable both for reading and for writing in units smaller than the block size, because then the blocks must be read into the buffer cache before they can be modified. The sequential access patterns are recognized based on a trace of the last 32 distinct blocks that were accessed. When a block B is accessed, this trace is scanned for blocks $B-1$, $B-2$, and so on up to $B-8$. The prefetch size is then defined to be the largest n such that blocks $B-1$ through $B-n$ were found in the trace. This causes blocks $B+1$ through $B+n$ to be prefetched. This mechanism increases the prefetching size up to a maximum of 512 KB depending on the length of the previously accessed contiguous data. It reduces the prefetching size if multiple independent files are being accessed, leading to conflicts in the buffer cache.

2.3 Experimental Setting

Vesta is implemented on an IBM SP1 platform. This is a distributed-memory MIMD machine. Each node is functionally equivalent to an IBM RS/6000 model 370 workstation, rated at 125 MFlops peak. The nodes are connected by a high-performance switch implemented as a multistage packet-switching network, with 40 MB/s duplex links [27]. The network adapters use programmed I/O rather than DMA, so heavy message passing activity comes at the expense of processing power. The network is synchronous across the machine, and provides a clock that is a globally synchronized to within about 0.1 μs.

The SP1 does not have dedicated I/O nodes. Instead, each compute node can be configured with either one or two IBM 0663 disks, which are used for paging and scratch space. Each disk has a capacity of 1 GB. The instantaneous transfer rate is 3.0 MB/s, but when various software and hardware overheads are taken into account (including sector and track overhead for ECC, bad sectors, SCSI command execution, etc.) this drops to about 2.26 MB/s for reads and 1.52 MB/s for writes. These numbers were measured using a test program that accessed an AIX logical volume using the same asynchronous I/O calls used by Vesta.

The installation we used for the experiments is a 16-node machine. Each node has 128 MB local memory and one disk. A 200 MB partition of the disk is configured as a special logical volume for Vesta. The rest is used for `/tmp` and swap space. In the experiments, we load the test program onto one subset of nodes, which assume the role of compute nodes. The Vesta server code is loaded onto the other nodes, which assume the role of I/O nodes. The Vesta servers use the dedicated logical volumes on their nodes for disk storage. The volumes on the other nodes are not used.

The system software consists of a full AIX running on each node. Message passing across the high-performance switch is provided by the EUI-H package (also known as the AIX Message Passing Library prototype/6000). EUI-H also provides a function to read the global clock. Loading and executing applications, including setting up the connection between the test program and the Vesta server, is done by the MPX package. This allows multiple clients partitions to connect to the same server.

Most of the presented results are based on a small number of measurements (or only one). This is adequate since we aim to characterize the system's maximal performance under deterministic conditions, rather than derive statistical measures for a random workload. In cases where multiple measurements were made, the best results are presented. This is legitimate because the reason for inconsistent measurements is usually a transient load created by other jobs on the system. Recall that each node runs a full AIX system, and we cannot prevent other processes from running on the nodes while we are running benchmarks. Therefore we simply discard inconsistent measurements if repeated measurements result in inconsistent values.

Finally, a note on terminology. When measuring rates, MB/s stands for one Million Bytes per second. When measuring file or block sizes, powers of two are used. Thus a 1 KB block is 1024 Bytes, and a 1 MB file is $2^{20} = 1048576$ Bytes. This may sound strange at first, but it is consistent with most published performance results.

3 Message-Passing Performance

The purpose of this experiment is to characterize the message-passing performance of Vesta, and the additional overhead above that to actually ship the data across the network. This is done by a set of measurements for different message sizes, spanning all powers of two from 16 bytes up to 256 KB. All these measurements are for one compute node and one I/O

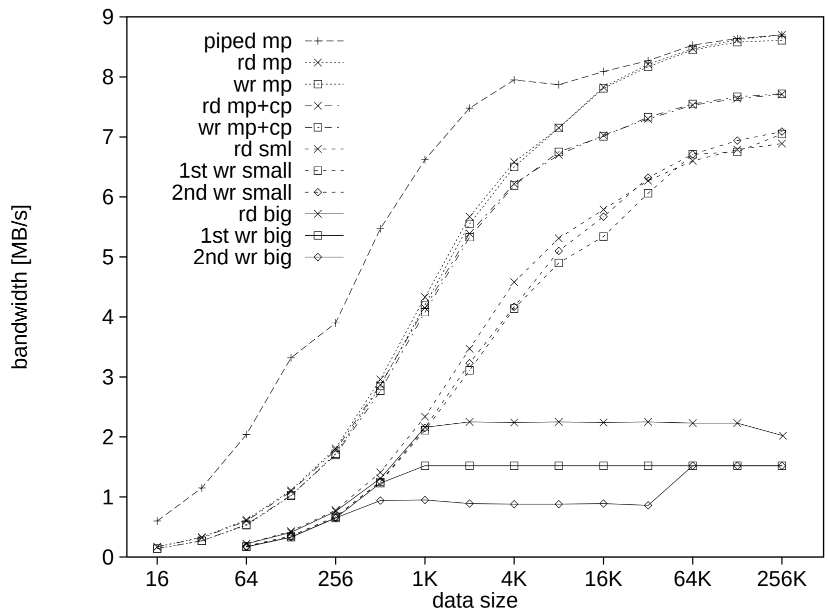


Figure 2: *Bandwidth as a function of data size, for message passing and Vesta.*

node. The results show that bandwidth improves with larger block sizes, until the network saturates (Fig. 2). Naturally, the bandwidth is much lower if we go to disk. The following description matches the order of the graphs in the figure from top-left to bottom-right.

The first set of measurements aims to characterize the message passing performance of the underlying communication subsystem (EUI-H). This consists of a loop to transfer 1000 messages between the two nodes, using the different message sizes. The loop is empty except for the message passing. In all cases, one node does blocking sends and the other does blocking receives. Note however that the EUI-H layer pipelines these messages: blocking implies that the data was copied out of the user's buffer, but not necessarily that it has been delivered at the other end. After the 1000 messages are sent, one empty message goes the other way to acknowledge that all the data has been received. The results show that a bandwidth of 8 MB/s is achieved for message sizes of 4 KB, and this goes up to 8.7 MB/s for the largest message sizes.

The second set aims to characterize Vesta message passing. This uses the same loop with 1000 iterations, but uses different message passing patterns that emulate Vesta read and write activity. For reads, there is a small constant-size request message, and then the data comes back with an acknowledgement in the same message (so the actual transfer is slightly larger than just the data). For writes, there is a small constant-size request message, then the data is sent in a second message, and finally an acknowledgement comes back. These patterns are exchanged between the two nodes, using the correct sizes for the requests and the acknowledgement, and the same data sizes as those used above. The results indicate that there is only a minor difference between the read pattern (two messages) and the write pattern (three messages). However, there is a significant difference between the back-and-forth patterns where each transfer is acknowledged, and the pipelined performance of EUI-H alone. As a result, the 8 MB/s mark is only approached when the data size is slightly above 16 KB.

Vesta message passing is further characterized by use of I/O vectors with 2-D elements. This allows multiple data elements that are not contiguous in either the cells or the subfile to be sent in a single message. However, if the underlying message passing system does not support these types of messages (and EUI-H does not), then an additional memory copy is required. The results show that this additional copy degrades performance for large data sizes, and limits the achievable bandwidth to about 7.7 MB/s. There is no measurable degradation for small sizes, because they are dominated by the startup cost, not the cost-per-byte.

All the above measurements simulated the Vesta message passing patterns without using actual Vesta code. The next set of measurements aims to characterize the performance of the full Vesta code path. This is done three times: for writes to unallocated space, for writes to pre-written space, and for reads. This kernel was chosen because overwriting existing data has different properties than writing to new space. The experiment involves a loop of synchronous reads or writes, using the same data sizes as before. The measurements were repeated twice, once for access to buffer cache (a total of 16 MB) and once for access to disk (a total of 128 MB).

The results for small files (which fit into memory) show that the Vesta code path further reduces the bandwidth, which reaches about 6.7 MB/s for access sizes of 64 KB, and peaks at over 7 MB/s for the largest size. This is largely due to copying the data into the buffer cache on the I/O nodes. As expected, access to large files is dominated by the overhead of accessing the disk. Writes achieve a bandwidth of about 1.5 MB/s, and reads a bandwidth of about 2.2 MB/s, both of which are the same as the bandwidths achieved by AIX-JFS. Given faster disks, we expect the achieved bandwidth to grow. It is very noticeable that the second write of blocks smaller than 64 KB achieves only about half the bandwidth of the first write. This is because the Vesta block size is 64 KB, and writing less than a block requires it to be read off the disk before being modified. This does not happen with small files, because the whole file is being cached.

Fig. 2 shows the bandwidth as a function of the data size. It is also possible to use the same measurements to plot the average time per I/O operation as a function of data size. This then allows us to assign costs directly to different components of the system.

The results are shown in Fig. 3. As may be expected, there is a linear relation between the time of an I/O operation and the amount of data being accessed. Therefore it makes sense to model the access time as $T = C + B \cdot n$, where C is a constant overhead per operation and B is a cost-per-byte. A least-squares fit leads to the values for C and B that are shown in Table I. It is seen that the Vesta message passing pattern suffers a higher startup cost than piped messages, due to the back-and-forth message passing for each operation. However, the startup cost for writes is essentially the same as for reads, despite the fact that they use an additional message. The additional copy for implementing the I/O vectors costs about another 0.013 μs per byte. Traversing the Vesta code path in access to small files leads to

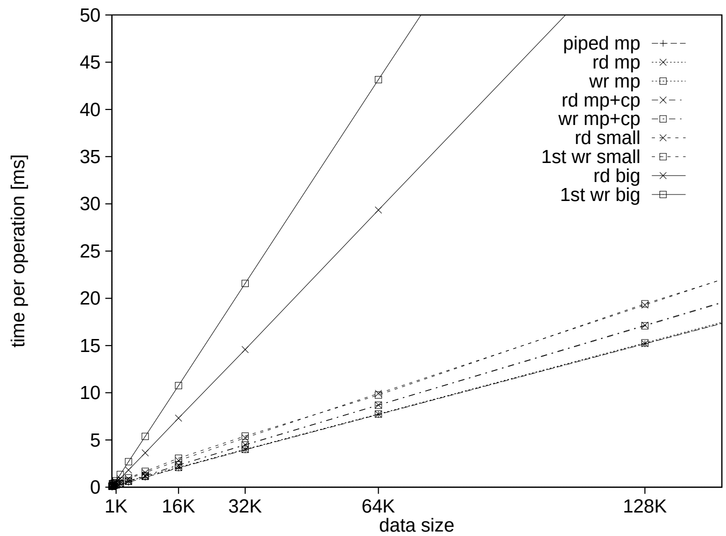


Figure 3: *Time per operation as a function of data size.*

operation	C	B
piped mp	0.067	0.000115
rd mp	0.143	0.000115
wr mp	0.143	0.000116
rd mp+cp	0.139	0.000129
wr mp+cp	0.143	0.000129
rd small	0.353	0.000144
1st wr small	0.489	0.000141
rd big	-1.04	0.000489
1st wr big	-0.02	0.000658

Table I: *Parameters in cost of operations, in ms.*

a higher startup cost, as expected; as it includes copying the data between system and user buffers, it also increases the cost-per-byte. When accessing disk, the cost-per-byte jumps by a factor of three to four because of the limited disk bandwidth. The startup cost seems to be negative, but this is an artifact of doing a least-squares fit of a line with a large slope and some data points with very large values: a far-out data point which is a bit above the line pushes the intersection with the y axis down.

4 Buffer Cache and Disk Synchronization

One of the effects of the buffer cache is that data can be stored in memory on the I/O nodes, without any access to the disks. This increases the effective bandwidth to that of memory transfer, as shown in the previous section. This is especially relevant for write operations,

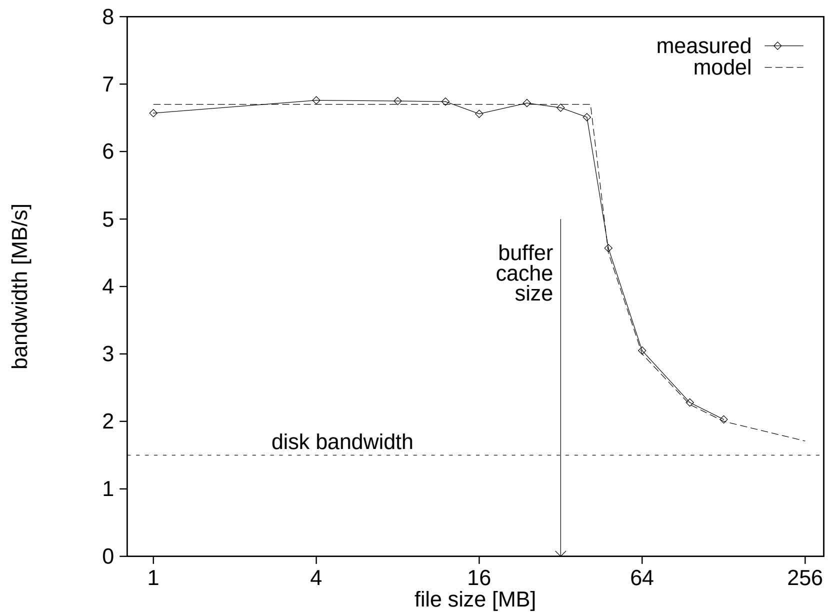


Figure 4: *Write bandwidth as a function of dataset size.*

because write-behind is always used, whereas reads must go to disk if the data is not already in the buffer cache. We therefore concentrate on write operations in this section.

Even when the total amount of data written is larger than the buffer cache, part of it may be left in memory while the rest is transferred to disk. The observed bandwidth will therefore be a weighted average of the memory and disk bandwidths. The model is that if the dataset size is less than or equal to the buffer cache size, it stays in memory. Only when the dataset size is larger, then we need to wait for part of it to be copied to disk to make room for the rest. Actually, writeback begins immediately, so the size where we have to wait is somewhat larger than the buffer cache. Denote by t the time to write the data. Assuming a memory transfer bandwidth of 6.7 MB/s, a disk bandwidth of 1.5 MB/s, and a buffer cache of 32 MB, the amount of data transferred out of the compute node is $t \times 6.7$, and the amount stored in the I/O node is $32 + t \times 1.5$. By equating the two we get $t = 6.15$ seconds, and the threshold dataset size is $t \times 6.7 = 41.23$ MB. Denoting the Dataset size by DS, the model for the observed bandwidth is therefore

$$BW = \begin{cases} 6.7 & \text{if } DS \leq 41.23\text{MB} \\ \\ \dfrac{DS}{6.15 + \frac{DS-41.23}{1.5}} & \text{otherwise} \end{cases}$$

This approaches the disk bandwidth of 1.5 MB/s for large datasets. As shown in Fig. 4, it is in excellent agreement with the measurements.

To avoid measuring the effect of deffered write-behind, all subsequent measurements include a call to the Vesta sync function, so that all data is actually transferred to disk.

Such calls were also used in the disk access measurements in the previous sections, but not in the buffer cache access measurements.

5 Parallel Access

The purpose of this experiment is to demonstrate and quantify the performance impact of parallel I/O. This is done by accessing different numbers of I/O nodes, and measuring the resulting bandwidth.

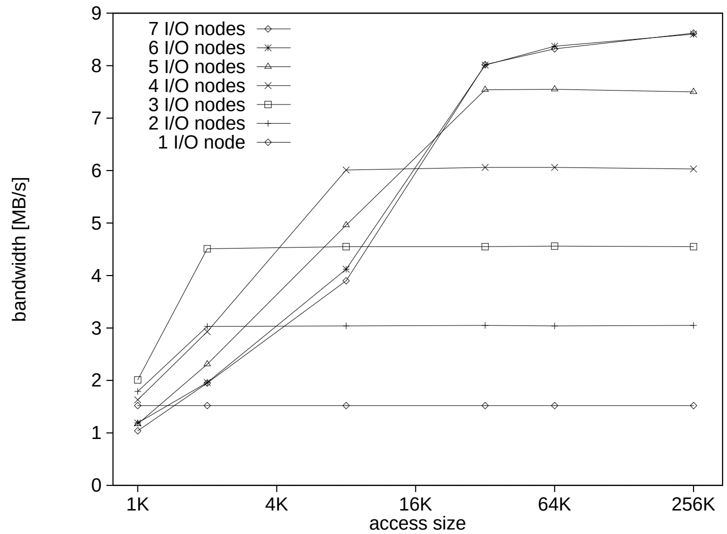
In order to allow parallel use of all the I/O nodes, a separate buffer is used to access each one, and the I/O operations are done asynchronously. The file size is also increased in proportion to the number of I/O nodes. The results are shown in Fig. 5. It is apparent that the bandwidth scales with the number of I/O nodes used, up to the limit set by the compute node's network port. For writes, an inversion is observed for small access sizes. The cause of this inversion is under investigation. In the second write, access sizes smaller than 64 KB achieve about 0.9 MB/s per I/O node accessed, because the data has to be read off disk before being modified. Above 64KB, writes achieve about 1.5 MB/s per I/O node accessed. Reads achieve about 2.1 MB/s per I/O node accessed. It is interesting to note that, given enough I/O nodes, writes actually achieve a higher bandwidth than reads. This might be because the compute node's network port is used more efficiently when data is being transmitted, because then the compute node has the initiative. When data is being received, messages from the different I/O nodes conflict when they converge on the compute node [2].

A drawback of this experiment is the limited bandwidth of the network port of the single compute node. This prevents checking scalability beyond about 6 I/O nodes. To address this issue, we conducted another experiment where the whole system was scaled. Based on the results in Fig. 5, we chose a ratio of 3 I/O nodes for each compute node. With this ratio, the port bandwidth is sufficient for both reads and writes. The results of this experiment are shown in Fig. 6, where an access size of 64 KB is used. The aggregate bandwidth of the system scales linearly with system size.

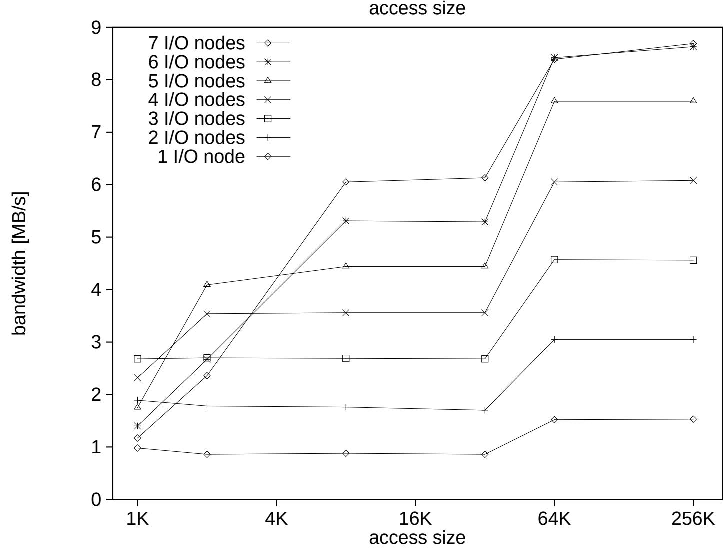
6 Orthogonal Logical Views

A major feature of Vesta is the ability to partition file data and access it in various ways. Two experiments were designed for the purpose of comparing the performance of different access patterns. The first is based on a single compute node that stripes data across cells in different ways. The second compares the performance of parallel access to subfiles that correspond to cells with the performance of access to subfiles that span multiple cells.

In the first experiment, one compute node and four I/O nodes are used. A file with one 128MB cell on each I/O node is created. The basic striping unit (BSU) is set to one fourth of the access size in each measurement, which changes from 256 bytes to 1 MB. Three access patterns are compared: using a striping unit of one BSU (so each access covers all four I/O nodes), using a striping unit of 4 BSUs (each access is contained in a single cell, 1st write



2nd write



read

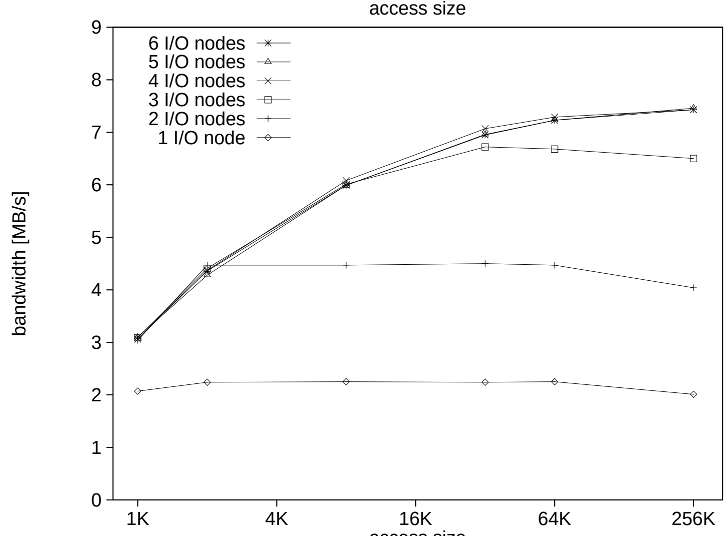


Figure 5: *Bandwidth as a function of access size and number of I/O nodes, for multiple asynchronous accesses.*

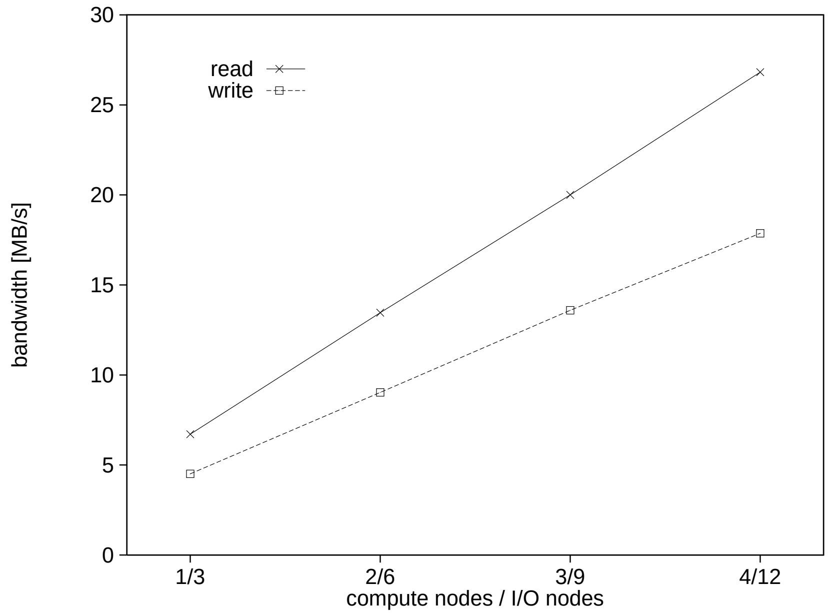


Figure 6: *Bandwidth as a function of system size, using a constant ratio of compute nodes to I/O nodes.*

but successive accesses hit all cells in a cyclic pattern), and using a striping unit of 128 MB (so each cell is completely accessed before the next one).

The results are shown in Fig. 7. The curves for writing are for the first write, to avoid the update effect for small sizes. When the cells are accessed sequentially, the bandwidth is essentially that of a single disk (write bandwidth is higher than 1.5 MB/s because we only sync to disk after writing the whole file, not after each cell). When each access is striped across all four I/O nodes, the bandwidth is quadrupled due to the parallelism.

The most interesting case is the cyclic pattern, where each access is to a different I/O node. Given that synchronous I/O operations were used, one might expect the bandwidth to be that of a single disk. However, the results indicate that the bandwidth is typically much higher. This is a result of the buffer cache management. Writes achieve a higher bandwidth due to the use of write-behind. This allows the actual disk access to be overlapped with access to subsequent I/O nodes, and effectively leads to parallel usage of the disks. Reads also achieve a higher bandwidth than a single disk, due to readahead. Again, this overlaps the prefetch disk operations with access to other I/O nodes, so when the data is actually requested it is already in memory. However, if the request is for more than the amount prefetched, the request is delayed until the missing data is obtained. This is the reason for the drop in performance when the access size is above 512 KB.

The second experiment is designed to investigate what happens when multiple compute nodes actually access disjoint subfiles simultaneously. This experiment is constructed as follows. Four compute nodes and four I/O nodes are used. A file with 4 cells is created, with one cell on each I/O node. The size of the file is 512 MB (i.e. 128 MB per I/O node).

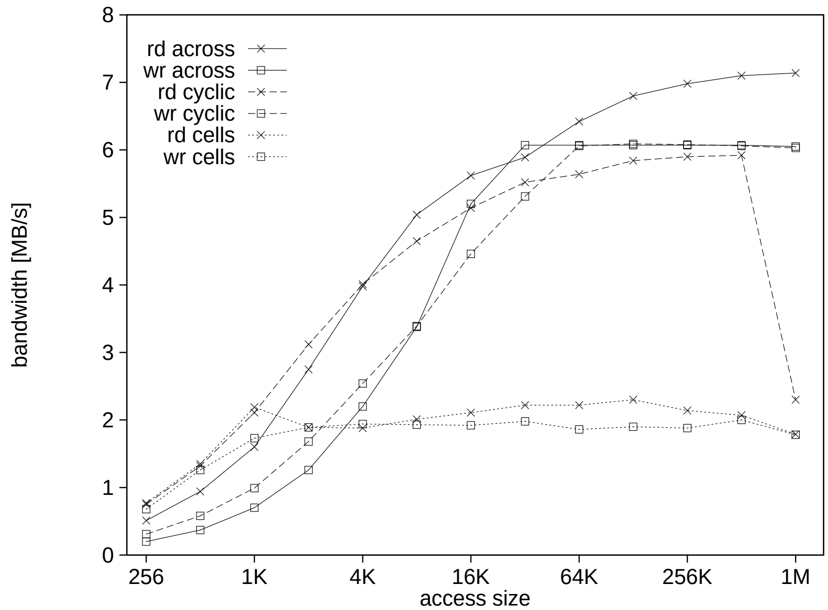


Figure 7: *Bandwidth as function of access size for different access patterns, with a single compute node and four I/O nodes.*

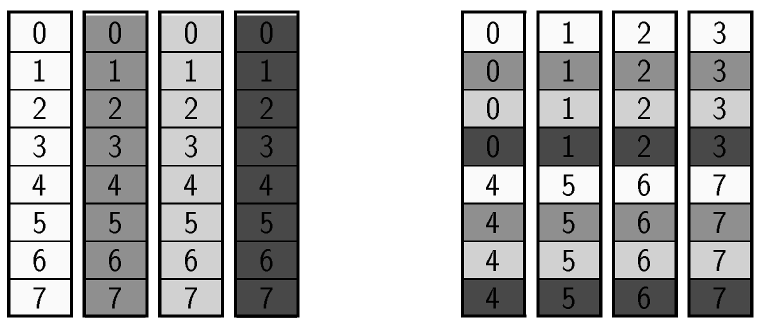


Figure 8: *Two ways to partition a Vesta file: subfiles correspond to cells, or they are striped across cells.*

The whole file is written twice and read, as in previous experiments, but in this case the file is first partitioned into four disjoint subfiles, and each compute node handles one of these subfiles. All four compute nodes synchronize at the end, to ensure that the measurement reflects the slowest compute node (this is equivalent to the "minimum sustained aggregate rate" in the terminology of French et al. [13]). The independent parameter is the access size used to write and read the file, which varies from 256 bytes through all powers of two up to 256KB. In each case the file is created with a basic striping unit (BSU) that is one fourth of the access size.

The pattern of writing and reading the file is repeated twice. In the first case, each subfile corresponds to a separate cell. In the second, subfiles are striped across cells (Fig. 8).

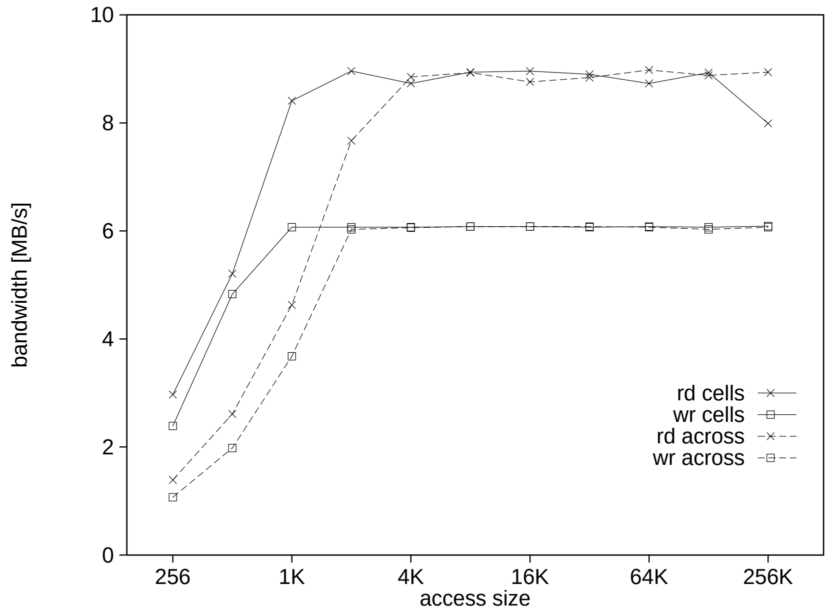


Figure 9: *Bandwidth as function of access size for different partitioning schemes, with four compute nodes and four I/O nodes.*

The results are shown in Fig. 9. When each compute node accesses a separate cell, the results are essentially the same as in Fig. 2 (one compute node to one I/O node) multiplied by 4. When the access is striped across cells, the effective access size to each cell is 1/4 the access size from the compute node (because each access is divided among the 4 cells). Therefore we would expect the bandwidth observed for accesses of b bytes using striping to be roughly the same as that for accessing $b/4$ bytes from a single cell. The results are actually better: when accessing b striped bytes, the observed bandwidth is that of accessing $b/2$ bytes from a single cell. This indicates that a large part of the overhead is global overhead for each operation, and does not depend on the data transfer size.

It is instructive to compare our results for Vesta with measurements done with other parallel file systems. The only other system that provides data decomposition like Vesta is the nCUBE system [8]. Detailed measurements of various access patterns are presented in [10]. These measurements indicate that when an access includes transposition (e.g. access by rows to data that is stored by columns) the degradation in performance can be very large. Their proposed solution is to use two-phase access. For example, a two-phase read is implemented by first accessing the disks in parallel and reading the data as it is stored, and then using message passing among the compute nodes to redistribute it as desired. Vesta achieves the same effect more directly, and without requiring extra buffers and copying on the compute nodes. Data is read off the disk and cached in the memory of the I/O nodes, and then it is redistributed when it is sent from the I/O nodes to the compute nodes.

Another problem in supporting data decomposition occurs when the accesses from the different processes are not part of the same collective I/O operation. In Vesta, processes define the subfile that they wish to access when it is opened, and then each process can access its subfile asynchronously, i.e. without coordination with other processes. As a result, requests may arrive at the I/O nodes in an arbitrary order. In particular, the order of requests may be different from the sequential order of data on the disk, leading to excessive seeking. Our results indicate that the Vesta buffer cache management algorithms are effective in overcoming out-of-order requests. Writes that arrive out of order are buffered and written later by the write-behind mechanism. Read requests that arrive out of order are nevertheless recognized as sequential in the aggregate, and thereby activate the prefetching mechanism. The prefetching reads the data off disk sequentially, so when the requests actually arrive they are satisfied from memory.

To summarize, our results indicate that the Vesta buffer cache management is an important contributor to performance. This corroborates independent studies that anticipated the advantages of caching based on analysis of applications [24, 17]. It also shows that with regular access patterns all processes benefit from the prefetching, thus refuting the anxiety raised in [18] regarding this issue.

7 Effectiveness of I/O Vectors

The purpose of this experiment is to evaluate the use of I/O vectors to combat fragmentation when data is interleaved at various granularities. Four compute nodes access different subfiles that are interleaved in a single I/O node. The access size is held constant at 64 KB. The interleaving is done in striping units that cover all powers of two between 16 bytes and 256 KB. The total file size (all four subfiles together) is 128 MB. As in other experiments, the whole file is written twice and then read. However, each compute node only accesses its subfile, and does not synchronize with the others in any way throughout the execution. There is a single synchronization point at the end, to ensure that the measurement reflects the last one to finish.

The results are shown in Fig. 10. They are compared with independent accesses from a single compute node to a single subfile (from Fig. 2). It turns out that the I/O vectors do indeed help to a certain degree. Specifically, the threshold where startup overhead begins to dominate the performance (and reduce the bandwidth) moves down from 512 bytes to 64 bytes. The drop in read performance above 64 KB is due to the fact that the access size is kept constant at 64 KB, and does not represent any problem with the I/O vectors. At these sizes, the combined accesses from the four compute nodes are larger than the prefetching window, so reading partial BSUs leads to some disk seeking.

8 Effectiveness of Asynchronous I/O

The purpose of this experiment is to evaluate the effectiveness of using asynchronous I/O to overlap I/Os with each other. While support for asynchronous I/O is not unique to Vesta, it was felt that this feature would be especially useful in a parallel file system.

The experiment involves a file with a cell on each of 8 I/O nodes. The file is accessed by

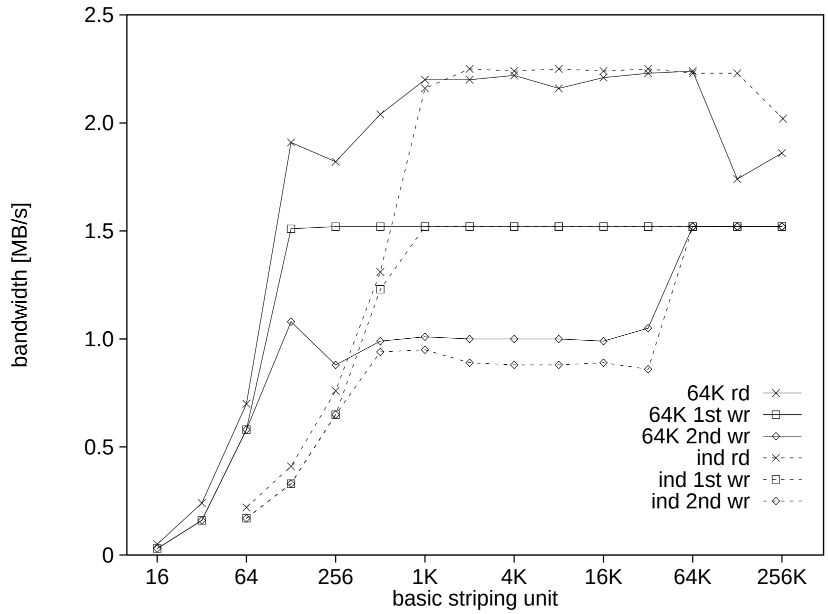


Figure 10: *Bandwidth as a function of striping unit, for interleaved access from four compute nodes to a single I/O node.*

a single compute node, using asynchronous I/O, with 1, 2, 4, 6, 8, 12, 16, and 20 buffers. The access size is 64 KB, and this is also the striping unit, so each access is fully contained in a distinct cell. The size of each cell is 128 MB. Both sequential and random access patterns were investigated.

The results are shown in Fig. 11. The results for sequential access are surprisingly high. Writes achieve the full network port bandwidth of about 8 MB/s for as few as two buffers. Reads saturate at about 7 MB/s, also with two buffers. These results are an outcome of the buffer cache management policies, as we already saw in Fig. 7. For writes, write behind is used, so the overhead is limited to copying the data in memory. The disk access itself is done later, and completes before the same node is hit again (recall that there are a total of 8 I/O nodes). Reads have to wait for the data to be fetched from disk, but the sequential access pattern allows prefetching to be used. Given the 2.2 MB/s disk rate, it is sufficient to keep three disks busy to saturate the network port into the compute node. The saturation value is lower for reads than for writes because of network conflicts when data converges on the compute node.

Random writes can achieve essentially the same performance as sequential writes, again by virtue of the write-behind mechanism. Random reads, on the other hand, cannot expect to benefit from prefetching. Therefore each read has to wait for a separate disk seek. Using 8 buffers is enough to keep all 8 I/O nodes busy, and achieves a large fraction of the possible bandwidth. Using more improves the performance somewhat by creating longer disk queues at the I/O nodes. With 20 buffers, the bandwidth is the same as for sequential access.

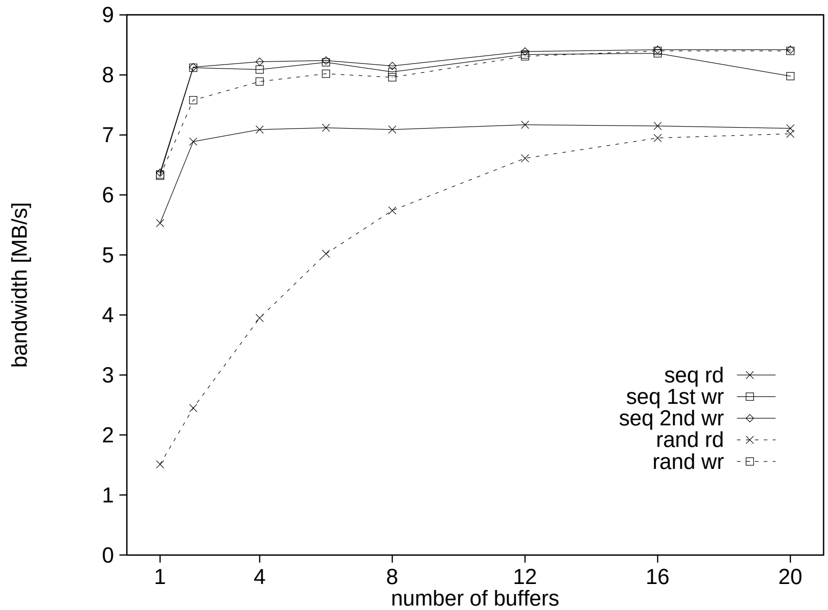


Figure 11: *Bandwidth as a function of the number of buffers used in asynchronous I/O.*

9 Performance of Sorting Application

Synthetic benchmarks that gauge a system's peak performance are important, but they do not provide a full picture. It is also important to investigate the degree to which applications can translate the performance figures into real benefits. We use an out-of-core sorting application as an example.

The sorting algorithm used is FastMeshSort [7]. This algorithm performs bitonic sorting iteratively along different dimensions of a mesh. It maps nicely onto Vesta, where subfiles that correspond to rows or columns of the data set can be opened and manipulated by independent compute nodes.

The results are shown in Fig. 12. The file being sorted had one million records. Four versions of the program were tested, with different sizes for the basic blocks of data that are loaded into memory for sorting. The smaller the block size, the more iterations that are needed to complete the sorting, and the more I/O intensive the application becomes. The case of 512 elements in the block is very I/O intensive, and exhibits excellent speedup as more I/O nodes are added. When larger block sizes are used, the speedup is somewhat smaller. Note that the speedups are taken relative to the execution with a single I/O node, for the same application.

While the speedup results are very promising, they don't tell the whole story. The total time required to sort the file is actually more important. This data is also shown in Fig. 12 (right), and indicates that the version with the largest block size (16K) is the most efficient. This means that for this specific application, it is better to use large block sizes, despite the fact that the speedup with added I/O nodes is then smaller. It does not mean that the parallel

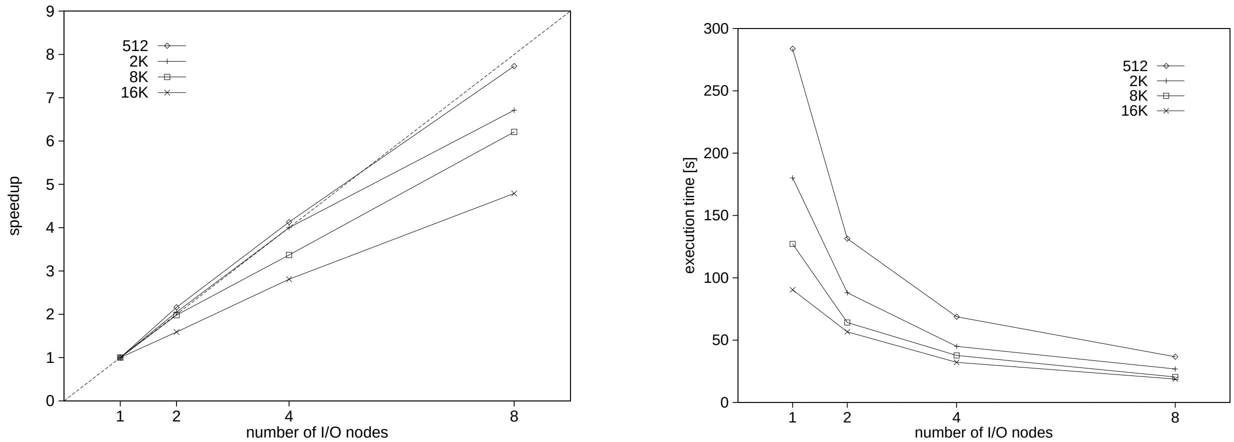


Figure 12: *Speedup and timing for FastMeshSort on 8 compute nodes and different numbers of I/O nodes.*

I/O provided by Vesta is useless; on the contrary, the results show that doubling the number of I/O nodes used provides a larger benefit than doubling the block size. Furthermore, it is not always possible to modify the compute-to-I/O ratio of an application, as it is done here by changing the block size. Parallel I/O can only improve the performance of the I/O component of an application. If this component is small, parallel I/O will not help, as a result of Amdahl's law. But if the I/O component is large, as it is for the 512-element blocks in FastMeshSort, then parallel I/O provides very significant benefits.

10 Conclusions

Parallel systems are rather complex, and even their designers cannot anticipate all the interactions among system components. Therefore detailed measurements are needed to understand system behavior. The measurements must be tailored to expose the unique characteristics of the system being studied. The results of such measurements can then be used to predict performance. We have conducted such measurements of the Vesta parallel file system, running on an IBM SP1 platform.

The results quantify the overhead of data transfer to and from the file system, both in terms of startup costs and in terms of cost per byte. It is shown that the system exhibits good scalability when the number of I/O nodes is increased. Moreover, access to subfiles that span multiple I/O nodes is essentially as efficient as access to independent I/O nodes. The results are expected to improve when Vesta is ported to new hardware and software platforms, notably new network adapters that use DMA and a communication subsystem that supports I/O vectors using the MPI interface on an SP2 computer. At the same time, we intend to run measurements on larger configurations and use additional tools, such as monitoring and tracing, to achieve a better understanding of the system.

Acknowledgements

Thanks to Won-Ho Chung who helped run some early performance studies in the summer of 1993. Thanks to Al Chang and Marsha Tennenhouse for their help in understanding the behavior of the original Vesta implementation, which was based on AIX-JFS. This analysis prompted the implementation of the Vesta buffer cache described here.

References

[1] R. Bordawekar, A. Choudhary, and J. M. del Rosario, "*An experimental performance evaluation of Touchstone Delta Concurrent File System*". In *Intl. Conf. Supercomputing*, pp. 367–376, 1993.

[2] E. A. Brewer and B. C. Kuszmaul, "*How to get good performance from the CM-5 data network*". In 8th *Intl. Parallel Processing Symp.*, pp. 858–867, Apr 1994.

[3] P. M. Chen and D. A. Patterson, "*Storage performance—metrics and benchmarks*". *Proc. IEEE* **81(8)**, pp. 1151–1165, Aug 1993.

[4] P. F. Corbett, S. J. Baylor, and D. G. Feitelson, "*Overview of the Vesta parallel file system*". In *Proc. IPPS '93 Workshop on I/O in Parallel Computer Systems*, pp. 1–16, Apr 1993. (Reprinted in *Comput. Arch. News* **21(5)**, pp. 7–14, Dec 1993).

[5] P. F. Corbett and D. G. Feitelson, "*Design and implementation of the Vesta parallel file system*". In *Scalable High-Performance Comput. Conf.*, pp. 63–70, May 1994.

[6] P. F. Corbett, D. G. Feitelson, J-P. Prost, and S. J. Baylor, "*Parallel access to files in the Vesta file system*". In *Supercomputing '93*, pp. 472–481, Nov 1993.

[7] P. F. Corbett and I. D. Scherson, "*Sorting in mesh connected multiprocessors*". *IEEE Trans. Parallel & Distributed Syst.* **3(5)**, pp. 626–632, Sep 1992.

[8] E. DeBenedictis and J. M. del Rosario, "*nCUBE parallel I/O software*". In 11th *Intl. Phoenix Conf. Computers & Communications*, pp. 117–124, Apr 1992.

[9] E. P. DeBenedictis and J. M. del Rosario, "*Modular scalable I/O*". *J. Parallel & Distributed Comput.* **17(1&2)**, pp. 122–128, Jan/Feb 1993.

[10] J. M. del Rosario, R. Bordawekar, and A. Choudhary, "*Improved parallel I/O via a two-phase run-time access strategy*". In *Proc. IPPS '93 Workshop on I/O in Parallel Computer Systems*, pp. 56–70, Apr 1993. (Reprinted in *Comput. Arch. News* **21(5)**, pp. 31–38, Dec 1993).

[11] P. C. Dibble, M. L. Scott, and C. S. Ellis, "*Bridge: a high-performance file system for parallel processors*". In 8th *Intl. Conf. Distributed Comput. Syst.*, pp. 154–161, 1988.

[12] D. G. Feitelson, P. F. Corbett, S. J. Baylor, and Y. Hsu, *Satisfying the I/O Requirements of Massively Parallel Supercomputers.* Research Report RC 19008 (83016), IBM T. J. Watson Research Center, Jul 1993.

[13] J. C. French, T. W. Pratt, and M. Das, "*Performance measurement of the Concurrent File System of the Intel iPSC/2 hypercube*". *J. Parallel & Distributed Comput.* **17(1&2)**, pp. 115–121, Jan/Feb 1993.

[14] M. Holland and G. A. Gibson, "*Parity declustering for continuous operation in redundant disk arrays*". In 5th *Intl. Conf. Architect. Support for Prog. Lang. & Operating Syst.*, pp. 23–35, Sep 1992.

[15] R. H. Katz, G. A. Gibson, and D. A. Patterson, "*Disk system architectures for high performance computing*". *Proc. IEEE* **77(12)**, pp. 1842–1858, Dec 1989.

[16] D. Kotz and C. S. Ellis, "*Caching and writeback policies in parallel file systems*". *J. Parallel & Distributed Comput.* **17(1&2)**, pp. 140–145, Jan/Feb 1993.

[17] D. Kotz and N. Nieuwejaar, "*Dynamic file-access characteristics of a production parallel scientific workload*". In *Supercomputing '94*, Nov 1994.

[18] D. F. Kotz and C. S. Ellis, "*Prefetching in file systems for MIMD multiprocessors*". *IEEE Trans. Parallel & Distributed Syst.* **1(2)**, pp. 218–230, Apr 1990.

[19] E. K. Lee and R. H. Katz, "*The performance of parity placements in disk arrays*". *IEEE Trans. Comput.* **42(6)**, pp. 651–664, Jun 1993.

[20] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S-W. Yang, and R. Zak, "*The network architecture of the Connection Machine CM-5*". In 4th *Symp. Parallel Algorithms & Architectures*, pp. 272–285, Jun 1992.

[21] D. B. Loveman, "*High Performance Fortran*". *IEEE Parallel & Distributed Technology* **1(1)**, pp. 25–42, Feb 1993.

[22] S. J. LoVerso, M. Isman, A. Nanopoulos, W. Nesheim, E. D. Milne, and R. Wheeler, "*sfs: a parallel file system for the CM-5*". In *Proc. Summer USENIX Conf.*, pp. 291–305, Jun 1993.

[23] J. Menon and D. Mattson, "*Comparison of sparing alternatives for disk arrays*". In 19th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 318–329, May 1992.

[24] E. L. Miller and R. H. Katz, "*Input/output behavior of supercomputing applications*". In *Supercomputing '91*, pp. 567–576, Nov 1991.

[25] B. Nitzberg, *Performance of the iPSC/860 Concurrent File System.* Technical Report RND-92-020, NASA Ames Research Center, Dec 1992.

[26] P. Pierce, "*A concurrent file system for a highly parallel mass storage subsystem*". In 4th *Conf. Hypercubes, Concurrent Comput., & Appl.*, vol. I, pp. 155–160, Mar 1989.

[27] C. B. Stunkel, D. G. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao, "*The SP1 high-performance switch*". In *Scalable High-Performance Comput. Conf.*, pp. 150–157, May 1994.