RC 19940 (02/21/95)
Computer Science/Mathematics

IBM Research Report

Parallel I/O Workload Characteristics Using Vesta

Sandra Johnson Baylor, C. Eric Wu
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden . T.J. Watson . Tokyo . Zurich

Research Report

Parallel I/O Workload Characteristics using Vesta

Sandra Johnson Baylor
C. Eric Wu

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents and will be distributed outside of IBM up to one year after the date indicated at the top of this page. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

Research Division
Almaden • T.J. Watson • Tokyo • Zurich

Parallel I/O Workload Characteristics using Vesta

Sandra Johnson Baylor
C. Eric Wu

IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10594

Abstract

In recent years, the design and performance evaluation of parallel processors has focused on the processor, memory and communication subsystems. As a result, these subsystems have better performance potential than the I/O subsystem. In fact, the I/O subsystem is the bottleneck in many machines. However, there are a number of studies currently underway to improve the design of parallel I/O subsystems. To develop optimal parallel I/O subsystem designs, one must have a thorough understanding of the workload characteristics of parallel I/O and its exploitation of the associated parallel file system. Presented are the results of a study conducted to analyze the parallel I/O workloads of several applications on a parallel processor using the Vesta parallel file system. Traces of the applications are obtained to collect system events, communication events, and parallel I/O events. The traces are then analyzed to determine workload characteristics. The results show I/O request rates on the order of hundreds of requests per second, a large majority of requests are for small amount of data (less than 1500 bytes), a few requests are for large amounts of data (on the order of megabytes), significant file sharing among processes within a job, and strong temporal, traditional spatial, and interprocess spatial locality.

I. Introduction

Many scientific applications written for parallel processors, particularly massively parallel processors, require high-bandwidth, low-latency access to I/O. However, in recent years, the design and performance evaluation of parallel processors has focused on the processor, memory, and communication subsystems. Also, while direct access storage devices have improved in recent years, the rate of improvement is far less than that for processors [1]. This has resulted in a large access gap between I/O accesses and processor and memory accesses; a factor that contributes to substantially better performance for processors and memory. To exploit the performance potential of a machine, a balance must exist between the processor, memory, communication, and I/O subsystems. However, the I/O subsystem is rapidly becoming a prohibitive bottleneck in many machines. To reduce or eliminate this bottleneck, more work is needed to improve the performance of these subsystems. There are a number of studies currently underway to improve the design of parallel I/O subsystems [2].

Also, the parallel I/O subsystems of many commercially available machines (such as the IBM SP2 [3], the Thinking Machines CM5 [4], the Intel Paragon [5], the nCUBE hypercubes [6], and the Tera [7]) allow data to be stored on the disks of several I/O nodes. This data may be accessed in parallel using a parallel file system such as the Vesta parallel file system [8], the IBM AIX Parallel I/O File System [9], the Intel Concurrent File System [10], and others [11, 12]. However, to develop optimal parallel I/O designs, one must have a thorough understanding of parallel I/O workload characteristics. To our knowledge, there is little information on this topic in the literature.

This paper presents the results of a study conducted to characterize the parallel I/O workloads of several scientific applications executing on a parallel processor. The machine consist of six heterogeneous nodes interconnected by a multistage interconnection network (see Section II for details). The applications access parallel files using the Vesta parallel file system [8, 13]. High-level parallel file system calls are traced, along with the associated lower-level message-passing calls. The traces are then analyzed to collect the various I/O statistics presented as seen at the application level. They include the I/O read and write request sizes, I/O request rates, the fraction of I/O reads and writes, and the amount of block sharing.

There are several commercial and experimental parallel file systems in existence. Many of them use the traditional sequential file system concepts with extensions for parallel file access. Also, many facilitate several modes of file access [14, 15]. These modes range from allowing completely decoupled accesses by the processes to providing the processes with synchronous access to the data, with or without data interleaving. Parallel file systems providing access to file data by various modes include the CM-5 CMMD [11], the Intel CFS [10], and the Express Cubix [16]. Other systems provide a memory mapped interface [17, 18]. Still other parallel file systems allow the user to specify different logical views and partitions of a file. This partitioning may or may not be independent of the system partitioning of the data. The Vesta parallel file system [8, 13] and the nCUBE I/O system [12] provide this feature. Other parallel file systems include PIFS (or Bridge) [19] which declusters blocks in round-robin fashion and

allows the file system to control which processor handles the data, and RAMA [20], which serves as a cache for a mass storage system with automatic file migration according to data usage.

The industrial and research communities have not defined a standard for parallel access to files. Until this occurs, parallel I/O workload characteristics are directly related to the parallel file system used by the application. In spite of this, it is believed that while collecting this information is beneficial for the design of the specific parallel file system used, it also provides a general understanding of how parallel files are accessed by applications. This information is useful in the design of high-performance parallel I/O architectures and of parallel systems. This information may also contribute to the standardization process.

There has been some work done on evaluating the I/O workload characteristics of applications. del Rosairo and Choudary [21] conducted an informal I/O analysis of several grand challenge problems. Miller and Katz [22] evaluated I/O intensive Cray applications, focusing on access rates. Pasquale and Polyzos [23, 24] analyzed the static and dynamic I/O characteristics of scientific Cray applications in a production workload. All of these studies evaluated sequential I/O workloads.

Parallel I/O applications may not exhibit the types of workloads found in traditional sequential applications; therefore they should be examined as well. Reddy and Banerjee [25] evaluated the I/O behavior of several Perfect benchmarks. The sequential benchmarks were parallelized and executed on an eight-processor Alliant. Only sequential access patterns were found; however, the I/O was not parallelized. Cypher, *et. al.*, evaluated the architectural requirements of parallel scientific applications. They measured the temporal patterns in I/O rates. These applications did not use a parallel file system to access files. In fact, the authors speculated that most of the applications were written to avoid the I/O bottleneck by using a data set size that could be placed in and accessed from primary memory. Kotz and Nils [26] evaluated the dynamic file-access characteristics of a parallel scientific workload. A production-based Intel iPSC/860 machine was used and the applications used the Concurrent File System [10] The results showed that parallelism resulted in smaller request sizes, a large amount of intra-job concurrent file sharing, non-consecutive sequential access and interprocess spatial locality. These results, while useful and interesting, are somewhat dependent upon the machine architecture and the file system used. More work is needed to determine the impact of the specific architectures and parallel file systems on the workload characteristics and to extract some basic parallel I/O workload characteristics that are independent of the architecture and file system used.

One method that can be used to exploit inherent parallelism of I/O in applications is to optimize the parallelism that can be extracted using the tools available from the parallel file system. This is not always intuitive and much work is needed in this area. This was not the focus of the work presented in this paper. It is assumed that the applications evaluated have already been optimized for parallelism.

The next section discusses the prototype machine used to run the parallel applications. It also describes the Vesta parallel file system used by the applications to obtain parallel file data access. Section III discusses the four applications evaluated and Section IV discusses the methodology used to obtain the parallel I/O workload characteristics. Section V presents the results and Section VI discusses the conclusions and suggests opportunities for future work.

II. Architecture and Parallel File System

The prototype machine used consists of 6 R/S 6000's (two model 530's and four model 550's) connected by a multistage interconnection network. This network is the same one used by the IBM SP1 and SP2 machines [27]. In addition to Vesta, a message passing library closely related to the MPL library provided by the SP2 is installed on the machine for communication. This library, known as MPX, includes all the features of the MPL library and an extension to provide for communication between parallel jobs in a client/server environment [9].

The Vesta parallel file system was designed to provide parallel file access to data from applications written for parallel processors. In Vesta, the traditional abstraction of a file consisting of a sequence of bytes is replaced with a new abstraction of a file as an entity that is partitionable into multiple disjoint sequences. Some user control over data layout is provided so that file data access can be tailored according to the anticipated access patterns of the application. For example, in a matrix multiply application, each compute node only has to access bands of rows or columns of the matrix. Using Vesta, each band can be placed on distinct I/O nodes. This can facilitate mutually exclusive access to each of the bands by the job tasks, reducing communication and processor interference overhead. Vesta also provides parallel access to files using several logical decompositions (or views) of the file. This enables applications to access different views of a file without physically moving the data. For example, a matrix multiply application may access bands of rows in one phase of the application, and then bands of columns in another phase without physical data movement.

Vesta also provides both blocking and non-blocking I/O commands. Blocking (or synchronous) commands do not complete until the entire I/O operation has completed. Non-blocking (or asynchronous) commands return immediately after data transfer is initiated. This allows useful computation to overlap the I/O access. A special function is provided to poll the status of the I/O operation or to wait for its completion. Vesta is implemented using a client/server methodology. Vesta libraries are called from a client (or compute) node. The actual file data is stored on a server (or I/O) node. The Vesta client and server processes communicate using the MPX library discussed earlier. More information on Vesta is found in [8, 13].

III. Applications

Four applications are used in this study. They are all written specifically for parallel processors using the Vesta parallel file system. The applications include a matrix multiply application, a sorting application, a seismic migration application, and a video server application.

Figure 1. File Data Placement for Matrix Multiply. (A) Each I/O Node Contains Two Cells, (B) Each Task Computes one Quadrant.

Matrix Multiplication

The matrix application multiplies two 1024x1024 element matrices and stores the result in a 1024x1024 element matrix. Each matrix element is a 4-byte floating point number. Also, each matrix is divided into four partitions with each one composed of 256 columns of the matrix. Each partition is allocated to a virtual I/O node (or cell). Cells are mapped to a physical I/O node in a round robin fashion beginning at a base I/O node that can be specified by the user. The data within each cell is stored in row-major order. Shown in Figure 1a is an illustration of this partitioning methodology.

This application ran on the prototype system with four compute nodes and two I/O nodes. Therefore, two cells were mapped to a single I/O node. For computational purposes, each matrix was divided into four blocks, each consisting of one quadrant of the matrix. The application job executing on the machine consisted of four tasks with each task executing on a single compute node and performing the matrix multiply on one quadrant of each of the matrices. Presented in Figure 1b is an illustration of this concept. Task 0, calculating elements for the top left quadrant of the resultant matrix, accesses file data from the top 512 rows of one matrix and the leftmost 512 columns of the other matrix. A similar process is used to calculate the elements for the remaining quadrants of the resultant matrix.

Sort

Four compute nodes and two I/O nodes are used in the sorting application. There are 4096 integers sorted and the numbers are grouped into blocks known as basic striping units (or BSUs). Each BSU is composed of 64 numbers (4 bytes each) or 256 bytes. The numbers are also decomposed into four cells of 1024 numbers or 16 BSUs. This application performs the sort in two phases. First, each compute node sorts a single cell using a recursive quicksort, followed by a final bubblesort. The compute nodes alternate the direction of the sort. For example, compute node 0 sorts in ascending order, compute node 1 sorts in descending order, etc. Second, a bitonic merge sort is performed on BSUs across the cells. This is done by initiating non-blocking **VESTA_READs** to BSUs, merging the data when the reads have completed, and then writing the results to the parallel file when the merge completes. This is an iterative process that completes when all merging is complete.

While the data set size for this application is small for a parallel application, the algorithm behavior for this application, including the access size, is independent of this size. As a result, it is believed that the behavior observed for this small data set size is similar to possible observed behaviors for larger data set sizes.

Seismic Migration

The seismic migration application uses implicit finite difference methods to perform 3D post-stack migration in the frequency-space domain. The seismic data, collected at various times, is stored as a file containing time-space domain data. It is then converted into the frequency-space domain using the fast fourier transform (FFT). This data is then stored as a frequency file in the Vesta file system. Since the speed of sound varies at different depths of the earth, an associated velocity correction file is used to store the velocity of the sound waves at specific depths. This velocity file is also stored as a Vesta file. This process is illustrated in Figure 2.

Both the frequency and velocity files consist of a 101x101 slice of data elements for each frequency and depth, respectively. Each data element of the frequency file is stored as an 8-byte (double) floating point number and each data element of the velocity file is stored as a 4-byte floating point number. The data for both files is arranged as a collection of BSUs arranged in a SxC array. S is the number of stripes (or slices) in the file, C is the number of cells, and the size of a BSU is $(\frac{(x \times y)}{nionodes}) \times 4$. In this application the number of cells equals the number of I/O nodes; four, with each cell physically stored on an I/O node. The BSU size is calculated as $(\frac{(101 \times 101)}{3}) \times 4 = 13600$ bytes.

Figure 2. Seismic Migration Methodology.

The seismic migration application has several phases of execution as shown in Figure 2. It is implemented as a manager/worker (or server/client) job. Several worker processes access concurrently disjoint slices of the Vesta frequency file. The manager process reads data from the velocity file, in parallel, for the first depth. It then broadcasts the data to the worker processes which in turn performs an extrapolation computation to obtain a corrected sub-image of that depth. The result is sent to the manager which calculates the corrected full image for that depth and stores this data in a depth file. (This file can either be a Vesta or other file system file). This process is then repeated for all depths. As an option, the manager can display different views of the depth file, as it is created, using visualization tools. More information on this application is found in [9].

The trace collected for this application includes the phases used by the manager and worker processes to read the frequency and velocity files and to perform their associated computations. The resulting depth file created was not a Vesta file; therefore, the trace did not include accesses to this file. Three compute nodes and three I/O nodes are used by this application. The seismic migration job consists of three processes; one manager process and two worker processes, with each process executing on a single compute node.

Video Server

In the video server application, compressed motion JPEG video files are striped across the I/O nodes in the prototype system. The application is written to maintain several continuous streams of compressed data, one per compute node, to video platforms that decode the compressed data and display it on attached televisions. This application consists of two types of processes executing concurrently on a node. First, the producer process reads a video packet of Vesta file data and places it in a shared buffer. Second, the consumer process reads the video packet from the shared buffer and then transfers it to the video platform. Mechanisms are provided to have the packets arrive at a constant rate to the video platform, to prevent the producer from overwriting the shared segment before the consumer can transfer the appropriate data, and to prevent the consumer from attempting to transfer data from an empty buffer.

Only the producer process is traced for this study. A 20 Mbyte compressed video Vesta file is read by this process. This file is created with 4 cells and with a BSU size of 1 byte. Each compressed video packet is 1454 bytes. Four I/O nodes (one for each cell of file data) and 2 compute nodes (one for each producer process) were used by the application to collect the trace. More information on the video server application is found in [28].

IV. Methodology

We used the Unified Trace Environment (UTE) to generate traces of the parallel applications. UTE, developed for the IBM SPn systems, facilitates the tracing of various events, such as message passing, parallel I/O, process dispatch, etc. [29]. Built on top of the AIX trace facility, UTE extends the trace facility for distributed software packages, such as MPL and its

associated MPX (message passing cross-partition) extension, and the Vesta file system. The goal of UTE was to provide a unified and easily expandable trace environment for performance measurement, analysis, and tuning. Without such a unified trace environment, tracing different software layers (e.g. messages passing routines, file system routines) may require multiple trace facilities and produce different and incompatible trace files. This not only makes trace generation more intrusive, but also makes performance analysis tedious and difficult.

Most trace systems require application source code modification to generate trace events. Thanks to the dynamic linking nature of the message passing layer, the trace facility requires no modification of application source code for trace generation. In this case, UTE traces the application from its beginning until it is finished. However, it is also possible to trace different phases of an application by inserting routines to turn tracing on and off in the code.

UTE inserts a couple of lines in each traced routine to cut trace records, one at the beginning of the routine (the b_ event) and the other at the exit of the routine (the e_ event). For each traced routine (from Vesta, MPX, etc.), the inserted code for trace generation is very similar:

```
routine_name(...)
{
    cut a begin event b_routine_name
    ....
    cut an end event e_routine_name
}
```

Each trace event generally consists of the event type, a timestamp, and any parameters passed to the event. For example, a b_Vesta_Read event consists of the b_Vesta_Read type, a timestamp, and the parameters passed to the Vesta_Read routine.

The trace facility pins the trace buffer in main memory to reduce the overhead associated with trace generation. The default buffer size is 128 Kbytes. When the buffer is full, events are written to the trace file. If the main memory on each node is large, the user may specify a larger trace buffer since this will reduce the trace generation overhead. Separate trace files are generated for each node in the system. A trace filename prefix is included as an option to UTE. This prefix is concatenated with a task number to form the name of the trace file in each node.

The trace facility uses the local system clock to generate timestamps. Each trace record has a 32-bit timestamp. If there is no event captured in a 0.9 second period, the trace facility automatically generates a heart beat event. This reduces the size of the timestamp from two words to one in each trace record. If a typical trace record has three words of data in addition to the hookword header, the average cost of cutting the trace record is around 110 machine instructions. Thus, the trace facility is very efficient and adds only a few microseconds for each trace event [30].

In addition to local clocks, IBM SPn systems provide a globally synchronized clock with three crystals, in which one is used as the master. This technique of triple modular redundancy (TMR) is used to guarantee a good global clock to drive the switch network. Each node in a SPn system has an adapter which connects the node to the switch network. The adapter includes a clock register driven by the master clock crystal. During system initialization the number of cycles required to set each clock register is calculated, and these registers are set one by one so that their values match one another. Since all clock registers are incremented by the same crystal, they are synchronized as one global clock.

UTE accesses the global clock periodically to minimize trace overhead. It accesses the SPn global clock before the user application starts, and generates a 4-word adapter clock record which includes both a double-word global clock value and a 32-bit AIX timestamp. After the application starts to run, UTE accesses the global clock once every 400 milliseconds to cut adapter clock records periodically. The difference in time between the global timestamp and local timestamp in an adapter clock record is then used in the analysis tools to adjust local timestamps, and merge multiple UTE/AIX trace files.

All applications are traced using the unified tracing environment (UTE). Separate traces are collected for each node used by the application, including the compute (or client) nodes and the I/O (or server) nodes. A UTE merge utility, **utemerge** is used to merge the multiple UTE/AIX trace files for the client and server traces separately. (Included in this paper are the results from the client traces only). The merged trace adds the processor identification number to the trace event. The merged trace stream is then passed to other tools for event listing and performance analysis. One of these tools is the utility, **lsute**, which lists and analyzes UTE/AIX trace files. This utility lists each event, including node ID, timestamp, and event name. As a result, parallel runtime information, such as the application run time and the message passing communication time, is derived. This utility also analyzes the trace and collects parallel I/O statistics such as the percentage of I/O reads and writes, including blocking and non-blocking accesses, the I/O request rates (interarrival times), the average time to process a request, the average and distribution of request sizes, and the amount of block sharing. The I/O interarrival time is calculated from the end of one I/O operation to the beginning of the next one. The I/O read and write percentages are calculated as a percentage of the total number of requests and of the total amount of data accessed by the application.

V. Results

Presented in this section are the results of the runtime execution analysis followed by the results of the parallel I/O analysis.

Partition of Total Elapsed Times

Since each node in an IBM SPn system may be shared by other processes, the information on how the total elapsed time is partitioned may be very useful. For the main process, the **lsute** utility shows both the time allocated for it and the time when the process is in its compute mode (i.e., not running in any message passing routine). The following tables show the time partition (all in seconds) for the four applications. For each application, the second column in each table shows the process ID associated with the application. The third column shows the total elapsed (trace) time, and the fourth column shows the time allocated for the main process. The last two columns indicate the time allocated for other processes and the main process compute time, respectively.

Table 1. Time Partition for Matrix.

Node	Main PID	Elapsed Time	Main Process	Other Processes	Compute Time
0	21207	739.991962	681.989961	58.002001	458.536395
1	13339	739.994536	731.223058	8.771478	727.249813
2	16386	739.994640	732.326021	7.668619	458.368027
3	14689	739.993578	735.494579	4.498999	458.353958

Table 2. Time Partition for Seismic

Node	Main PID	Elapsed Time	Main Process	Other Processes	Compute Time
0	20843	282.158459	229.960944	52.197515	3.176713
1	16346	282.087362	275.849028	6.238335	233.876249
2	14806	282.078737	275.403514	6.675223	237.815924

Table 3. Time Partition for Sort.

Node	Main PID	Elapsed Time	Main Process	Other Processes	Compute Time
0	26554	2.905508	2.050845	0.854664	0.827038
1	12993	2.199644	1.989975	0.209669	1.290612
2	14767	2.179864	2.054348	0.125516	0.811922
3	15117	2.136253	1.993837	0.142416	1.224270

Table 4. Time Partition for Video.

Node	Main PID	Elapsed Time	Main Process	Other Processes	Compute Time
0	12132	58.839355	38.512746	20.326609	11.216337
1	12066	61.504248	38.656303	22.847945	15.786958

Since we have 4 IBM RS/6000 model 550's and 2 RS/6000 model 530's, it is expected that some nodes are running slower than others, depending on what nodes are chosen at run time. For example, node 1 in the Matrix application and both node 1 and node 3 in the Sort application are slower models. Both applications have the amount of computations evenly distributed among four nodes. Therefore, the compute time in these slower nodes are longer than those in other nodes executing the application.

Histogram of Routines

The **lsute** utility is able to report a histogram of message passing and Vesta I/O routines, including the total elapsed time in each routine and counts for each node. For each call to a message passing or I/O routine, the elapsed time is calculated from the begin event to the end event, including periods stolen by other processes, if any. Table 5 shows the histogram for the matrix multiply application.

The histogram of routines is very useful to identify bottlenecks. For example, because node 1 (model 530) is slower than other nodes, all other nodes end up spending more than 230 seconds in order to reach a synchronization point (using **MP_SYNC()**) with node 1. If node 1 was also a model 550, the total elapsed time would be substantially improved. Also, observe that the times to do I/O operations on the faster machines are 88-90 microseconds (us) for **VESTA_OPEN**, 9.5 us for **VESTA_CLOSE**, 800-900 milliseconds (ms) for **VESTA_READ**, and 1.9 seconds for **VESTA_WRITE**.

As another example, Table 6 shows a partial routine histogram for the seismic migration application. Observe that the time for the worker nodes (1 and 2) to do a **VESTA_READ** ranges from 578 to 605 ms. It can be seen that node 0 in the Seismic application spends too much time (239 seconds in **MP_REDUCE()**) to collect data from other nodes. This shows that the manager/worker model used in this application results in the minimal use of the compute node by the manager process. A redesign of this application to exploit the computing power of all nodes may result in a more efficient solution.

Included in Table 7 is a partial routine histogram for the sort application. This table shows that a large number of **MP_STATUS** routines have been called. This is because the application is written to continuously check to determine if outstanding **VESTA_READs** have completed before proceeding to merge and to determine if outstanding **VESTA_WRITEs** have completed before starting another bitonic merge iteration. The typical times to complete the I/O operations for this application are 109 to 176 us for **VESTA_OPEN**, 133 to 199 us for **VESTA_CLOSE**, 427 to 685 us for **VESTA_READ**, and 511 to 822 us for **VESTA_WRITE**.

Presented in Table 8 is a partial routine histogram for the video server application. The times to execute the I/O operations range from 142 to 232 us for **VESTA_OPEN**, from 15 to 25 us for **VESTA_CLOSE**, and from 2.8 to 3.2 ms for **VESTA_READ**.

Table 5. Histogram for Matrix Multiply Application.

		Total_elapsed_time	Count	Average
72 MP_WAIT				
	node 0	3.724144640	18	0.206896924
	node 1	3.849591296	18	0.213866183
	node 2	4.139567104	18	0.229975950
	node 3	3.633470464	18	0.201859470
45 MPX_BSEND				
	node 0	0.000454912	6	0.000075819
	node 1	0.001527296	13	0.000117484
	node 2	0.001073920	14	0.000076709
	node 3	0.000973312	12	0.000081109
45 MPX_BRECV				
	node 0	0.157442560	6	0.026240427
	node 1	0.155347712	13	0.011949824
	node 2	0.156630272	14	0.011187877
	node 3	0.157015552	12	0.013084629
40 MPX_SEND				
	node 0	0.053945856	10	0.005394586
	node 1	0.001049344	10	0.000104934
	node 2	0.000716032	10	0.000071603
	node 3	0.000723968	10	0.000072397
32 MPX_RECV				
	node 0	0.030031104	8	0.003753888
	node 1	0.000179968	8	0.000022496
	node 2	0.000113408	8	0.000014176
	node 3	0.000112640	8	0.000014080
12 VST_Open				
	node 0	0.000265216	3	0.000088405
	node 1	0.000416000	3	0.000138667
	node 2	0.000264192	3	0.000088064
	node 3	0.000270080	3	0.000090027
12 VST_Close				
	node 0	0.000028416	3	0.000009472
	node 1	0.000045056	3	0.000015019
	node 2	0.000028928	3	0.000009643
	node 3	0.000028672	3	0.000009557
8 VST_Read				
	node 0	1.874770224	2	0.937385112
	node 1	2.409436747	2	1.204718373
	node 2	1.621896249	2	0.810948125
	node 3	2.113450651	2	1.056725325
4 MP_SYNC				
	node 0	238.736355785	1	238.736355785
	node 1	0.000598784	1	0.000598784
	node 2	272.492288512	1	272.492288512
	node 3	274.979738624	1	274.979738624
4 VST_Write				
	node 0	2.293882573	1	2.293882573
	node 1	1.922744359	1	1.922744359
	node 2	2.814739874	1	2.814739874
	node 3	1.819664928	1	1.819664928

Table 6. Histogram for Seismic Migration Application.

		Total_elapsed_time	Count	Average
612 MP_WAIT				
	node 0	1.117975040	600	0.001863292
	node 1	0.386056704	6	0.064342784
	node 2	0.390024192	6	0.065004032
306 MPX_RECV				
	node 0	0.055998976	300	0.000186663
	node 1	0.000070656	3	0.000023552
	node 2	0.000065792	3	0.000021931
306 MPX_SEND				
	node 0	0.037341696	300	0.000124472
	node 1	0.000534272	3	0.000178091
	node 2	0.000547072	3	0.000182357
303 MP_BCAST				
	node 0	5.259245014	101	0.052071733
	node 1	39.808331008	101	0.394141891
	node 2	34.313304320	101	0.339735686
297 MP_REDUCE				
	node 0	239.496146818	99	2.419152998
	node 1	0.912578048	99	0.009217960
	node 2	1.611256133	99	0.016275314
102 VST_Read				
	node 0	1.624692327	100	0.016246923
	node 1	0.578206636	1	0.578206636
	node 2	0.605115116	1	0.605115116
22 MPX_BRECV				
	node 0	0.077114112	14	0.005508151
	node 1	0.427805952	4	0.106951488
	node 2	0.869400576	4	0.217350144
22 MPX_BSEND				
	node 0	0.001617664	14	0.000115547
	node 1	0.000937216	4	0.000234304
	node 2	0.000974336	4	0.000243584
9 MP_SYNC				
	node 0	0.052627968	3	0.017542656
	node 1	0.052929792	3	0.017643264
	node 2	0.023550720	3	0.007850240
4 MP_BSEND				
	node 0	0.000233472	2	0.000116736
	node 1	0.000104960	1	0.000104960
	node 2	0.000104448	1	0.000104448
4 MP_BRECV				
	node 0	0.000185344	2	0.000092672
	node 1	0.563749632	1	0.563749632
	node 2	0.566674944	1	0.566674944

Table 7. Histogram for Sort Application.

	Total_elapsed_time	Count	Average
63420 MP_STATUS			
node 0	0.174584320	13855	0.000012601
node 1	0.268841472	13442	0.000020000
node 2	0.188825600	11529	0.000016378
node 3	0.361508096	24594	0.000014699
4416 MPX_SEND			
node 0	0.098186240	1104	0.000088937
node 1	0.146408192	1104	0.000132616
node 2	0.100147200	1104	0.000090713
node 3	0.090739200	1104	0.000082191
2944 MPX_RECV			
node 0	0.011414272	736	0.000015509
node 1	0.019034624	736	0.000025862
node 2	0.011376896	736	0.000015458
node 3	0.011533312	736	0.000015670
1472 VST_Write_Q			
node 0	0.198030848	368	0.000538127
node 1	0.302560056	368	0.000822174
node 2	0.194181120	368	0.000527666
node 3	0.188407040	368	0.000511976
1472 VST_Read_Q			
node 0	0.161702460	368	0.000439409
node 1	0.252081664	368	0.000685005
node 2	0.162566249	368	0.000441756
node 3	0.157148928	368	0.000427035
128 MP_WAIT			
node 0	0.000570112	32	0.000017816
node 1	0.012742400	32	0.000398200
node 2	0.000539904	32	0.000016872
node 3	0.030774784	32	0.000961712
20 MP_SYNC			
node 0	0.684324503	5	0.136864901
node 1	0.004809984	5	0.000961997
node 2	0.689521552	5	0.137904310
node 3	0.046818304	5	0.009363661
8 VST_Close			
node 0	0.000026368	2	0.000013184
node 1	0.000039936	2	0.000019968
node 2	0.000026624	2	0.000013312
node 3	0.000026880	2	0.000013440
7 MPX_BRECV			
node 0	0.262105344	4	0.065526336
node 1	0.261712384	1	0.261712384
node 2	0.264406016	1	0.264406016
node 3	0.265986560	1	0.265986560
7 MPX_BSEND			
node 0	0.000296448	4	0.000074112
node 1	0.000121856	1	0.000121856
node 2	0.000081664	1	0.000081664
node 3	0.000082944	1	0.000082944
4 VST_Open			
node 0	0.000113664	1	0.000113664
node 1	0.000176128	1	0.000176128
node 2	0.000109056	1	0.000109056
node 3	0.000110336	1	0.000110336

Table 8. Histogram for Video Server Application.

		Total_elapsed_time	Count	Average
55028	MP_WAIT			
	node 0	29.287130112	27514	0.001064445
	node 1	24.866950144	27514	0.000903793
27514	MPX_RECV			
	node 0	0.310622976	13757	0.000022579
	node 1	0.465056000	13757	0.000033805
27514	MPX_SEND			
	node 0	1.463526400	13757	0.000106384
	node 1	2.592256000	13757	0.000188432
27514	VST_Read			
	node 0	39.519978654	13757	0.002872718
	node 1	44.969038335	13757	0.003268811
4	MPX_BSEND			
	node 0	0.000764672	2	0.000382336
	node 1	0.001314816	2	0.000657408
4	MPX_BRECV			
	node 0	0.139878912	2	0.069939456
	node 1	0.072524032	2	0.036262016
2	VST_Close			
	node 0	0.000015872	1	0.000015872
	node 1	0.000025088	1	0.000025088
2	MPX_ATTACH			
	node 0	1.854062848	1	1.854062848
	node 1	0.623672064	1	0.623672064
2	VST_Attach			
	node 0	0.085728000	1	0.085728000
	node 1	0.035142400	1	0.035142400
2	VST_Init			
	node 0	1.945437282	1	1.945437282
	node 1	0.766006827	1	0.766006827
2	VST_Open			
	node 0	0.000142080	1	0.000142080
	node 1	0.000232960	1	0.000232960

I/O Statistics

Presented in Tables 9 and 10 are the I/O access counts and the I/O request sizes, respectively, or the four applications. Table 9 shows that the number of Vesta I/O events range from 12 to over thirteen thousand, with the percentages of **VESTA_READs** ranging from 50% to 100%. Also, two applications, matrix and seismic, use only blocking Vesta I/O calls and the remaining applications use only non-blocking calls. As shown in Table 10, the size of a request varies from 256 bytes to over 1.7 MBytes and the percentages of the total bytes accessed that are reads vary from 50% to 100%. For the matrix application, comparing this percentage with that of the percentage presented in Table 9 shows that a larger amount of data was requested per **VESTA_READ**. For all of these applications, the request sizes, per application, are generally constant.

An examination of Tables 9 and 10 together shows that the overwhelming majority of the requests are for small amounts of data (less than 1500 bytes). Also, a relatively small number of requests are for request sizes for 1.6 to 1.7 megabytes. This result is consistent with a pre-

vious I/O study that observed similar behavior [26]. (This study is referred to hereafter as the Kotz and Nils study).

Table 5. I/O Access Counts.

CODE	RDS	BRDS	NBR	WRTS	BWR	NBWR	TOTAL	%R	%W
Matrix	8	8	0	4	4	0	12	66.7	33.3
Seismic	102	102	0	0	0	0	102	100.0	0.0
Sort	1472	0	1472	1472	0	1472	2944	50.0	50.0
Video	13757	0	13757	0	0	0	13757	100.0	0.0

Table 6. I/O Request Sizes.

CODE	AVG	AVG-R	AVG-W	%RDS	%WRTS
Matrix	1747626	2097152	1048576	80.0	20.0
Seismic	1632160	1632160	0	100.0	0.0
Sort	256	256	256	50.0	50.0
Video	1454	1454	0	100.0	0.0

Illustrated in Tables 11 and 12 are the average interarrival times (in ms) and request rates (in requests/sec) for the sort and video server applications, respectively. The numbers are also given for reads and writes for sort. The video server application only issued **VESTA_READ** calls, so the results are presented for reads. The interarrival times and request rates are presented for each compute node used by the application and also for the average. The interarrival times range from 1.69 to 2 ms, with an average of 1.86 for sort, and from 3.15 to 3.47 ms, with an average of 3.31 ms for the video server. This translates to an average request rate of 542 and 303 requests/second for sort and the video server, respectively. To our knowledge, these numbers are the first to present request rates for parallel I/O usage. The results of more applications are needed for Vesta and other file systems to determine general request rates, if any, for parallel I/O. These rates can then be used to evaluate, improve, and develop new parallel I/O designs [2].

Table 7. I/O Interarrival Times (IAT, ms) andd Request Rates (RQR, rq/sec) for Binary Sort.

NODE	IAT	R-IAT	W-IAT	RQR	R-RQR	W-RQR
0	1.88	4.21	4.15	533	237	241
1	1.69	4.09	3.97	590	244	252
2	1.87	4.20	4.15	534	238	241
3	2.00	4.31	4.27	510	232	234
Average	1.86	4.20	4.12	542	238	242

Table 8. I/O Interarrival Times (IAT, ms) and Request Rates (RQR, rq/sec) for Video Server Application.

NODE	R-IAT	R-RQR
0	3.47	288
1	3.15	317
Average	3.31	303

It is intuitive to think that processes within a parallel job will share files. This behavior is in fact observed for the applications considered in this study. It is more interesting to observe the sharing behavior at the block level. The I/O analysis portion of **lsute** calculated the access patterns, including the read and write counts and the interarrival times, at the block level. A block is determined by first calculating the cell within a block and then the block within a cell for each access. The block size is determined by the user of the **lsute** utility and in this study, the block size varies from 32 to 512 bytes for the sort application, and from 4K to 64K bytes for the remaining applications. For all of the applications, accesses to each block are regular and consistent, *i.e.*, approximately the same number of accesses were made to each block, independent of the block size. It is believed that this is a result of the applications used and of the fact that Vesta allows the user to decompose file data to facilitate mutually exclusive regular accesses for a wide range of applications.

Also, the blocks are shared by all compute nodes for sort and for the video server, and by two of the four compute nodes for matrix. The sort and matrix application exhibited read/write sharing at the block level and the video server exhibited read-only sharing. No blocks are shared in the seismic application. (It was the desire of the seismic programmer to achieve this result, as facilitated by Vesta. The **lsute** parallel I/O analysis option allowed the programmer to verify this). This block sharing observation, denoted interprocess spatial locality by Kotz and Nils, is very different from the results obtained from traditional and distributed systems workloads where it is uncommon to share files and where read/write sharing is infrequent [31]. However, the result is consistent with the Kotz and Nils study, where interprocess spatial locality was observed.

Figure 3. Block Interarrival times for Sort. Left-Average. Right-Per Compute Node.

A histogram of block interarrival times for sort are presented in Figure 3. Figure 3a presents the overall block results and Figure 3b presents the results for each compute node accessing the block. The results show the overwhelming majority of the interarrival times for the block are less than 5 ms, with over 78% less than 1 ms. Most of the blocks accessed by this application and by the matrix application exhibit similar behavior. Also, most of the blocks accessed are stored sequentially in the cell. This shows strong locality, both temporal and spatial. The sequential block access result is in contrast to the non-consecutive sequential accesses in the Kotz and Nils study. This is a direct result of the utilization of the Vesta feature giving the user control over data layout. This feature was not available in the Intel CFS, the parallel file system used in the Kotz and Nils study.

The locality results suggest that caching can be used to optimize performance for these applications. Since there is significant read/write sharing at the compute node, caching there may not be desirable due to consistency constraints. However, a disk cache at the I/O node can be used to optimize parallel I/O performance. The remaining two applications consisted primarily of large sequential accesses with little reuse of data. Therefore, they would not benefit from the use of a disk cache.

VI. Conclusion

This study was conducted to evaluate the workload characteristics of parallel applications using Vesta. Four parallel applications, executing on a prototype machine consisting of a collection of RS/6000s workstations interconnected by a SPn switch, were traced and analyzed for program execution and parallel I/O characteristics. The results presented show that applications can decompose file data into sub-units, facilitating its mutually exclusive parallel access. Also, a large majority of the I/O requests are for small amounts of data, a few requests are for

large amounts of data, there is significant file sharing among processes within a job, and there is widespread interprocess locality. These observations are consistent with those of the Kotz and Nils parallel I/O study [26]. Since that study used a different architecture (the Intel iPSC/860) and a different parallel file system (the Intel CFS), these results may be characteristic of parallel I/O in general. More studies are needed to confirm this assertion.

Furthermore, we observed traditional spatial locality to blocks within a Vesta cell (sub-file). This is in contrast to the Kotz and Nils study, where non-consecutive sequential accesses were observed. This contrast is attributed to the Vesta feature that allows users to specify file data placement. This feature is not available in CFS. The temporal, traditional spatial and interprocess spatial locality results suggest that disk cache design optimizations can be used on the I/O nodes of parallel I/O subsystems. More results are needed from a wider range of parallel applications before such a design can be fully developed and before more general conclusions are drawn.

The tracing and analysis tools have been modified for use with the IBM parallel I/O file system (PIOFS) [9]. This file system is designed for the SPn computers; therefore, more analysis can be done on a spectrum of parallel applications to broaden the scope of the results presented here and to contribute to the formulation of general characteristics of parallel I/O workloads. These tools may also be used to study the various parameters associated with PIOFS to optimize file data placement and access patterns for optimal application execution.

Acknowledgements

We thank George Almasi, Peter Corbett, Dror Feitelson, and Jean-Pierre Prost for providing us with the parallel applications used in this evaluation. Also, we thank them for discussing the algorithms used and the design of the applications; providing us with insight for interpreting the results. We thank George Almasi for providing us with Figure 2, the seismic migration methodology. We also thank Caroline Benveniste and Yarsun Hsu for helpful suggestions and discussions regarding the composition of the analysis tools.

REFERENCES

[1] Katz, R. H., G. A. Gibson, and D. A. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE*, vol. 77, no. 12, pp. 1842-1858, December, 1989.

[2] Baylor, S. J. and C. E. Wu, "Parallel I/O Workload Characteristics Using Vesta," *Submitted to the Third Annual IPPS Workshop on I/O in Parallel and Distributed Systems*, pp. 1-18, April, 1995.

[3] International Business Machines Corporation, Scalable POWERparallel Systems High-Performance Technical Computing Solutions, Publication # GH23-2485-00, March 1994.

[4] Leiserson, C.E., et. al., "The Network Architecture of the Connection Machine CM-5," *4th Symposium on Parallel Algorithms and Architectures*, pp. 272-285, June 1992.

[5] Messina, P., "The Concurrent Supercomputing Consortium: Year 1," *IEEE Parallel and Distributed Technology*, vol. 1, no. 1, pp. 9-16, February 1993.

[6] Hayes, J.P., Mudge, T., Stout, Q.F., Colley, S., and Palmer, J., "A Microprocessor-based Hypercube Supercomputer," *IEEE Micro*, vol. 6, no. 5, pp. 6-17, October 1986.

[7] Alverson, R., Callahan, D., Cummings, B., Koblenz, B., Porterfield, A., and Smith, B., "The Tera Computer System," *International Conference on Supercomputing*, pp. 1-6, June 1990.

[8] Corbett, P.F., and Feitelson, D.G.,, "Design and Implementation of the Vesta Parallel File System," *Proceedings of the Scalable High Performance Computing Conference*, pp. 63-70, 1994.

[9] Corbett, P.F., Feitelson, D.G., Prost, J.-P., Almasi, G., Baylor, S.J.,, Bolmarcich, A.S., Hsu, Y., Satran, J., Snir, M., Chi, M. and Colao, R., Herr, B., Kavaky, J., Morgan, T.R., and Zlotek A., "Parallel File Systems for the IBM SP Computers," *To appear in the IBM System Journal*, pp. 1-25, 1995.

[10] Pierce, P., "A Concurrent File System for a Highly Parallel Mass Storage Subsystem," *Proceedings of the Fourth Conference on Hypercube Computers and Applications*, pp. 155-160, 1989.

[11] Best, M.L., Greengerg, A., Stanfill, C., and Tucker, L.W., "CMMD I/O: A Parallel Unix I/O," *Proceedings of the Seventh International Parallel Processing Symposium*, pp. 489-495, April 1993.

[12] DeBenedictis, E. and del Rosario, J. M., "nCUBE Parallel I/O Software," *Proceedings of the 1992 International Phoenix Conference on Computers and Communicaitons*, pp. 117-124, April 1992.

[13] Corbett, P.F., D.G. Feitelson, J-P. Prost, and S.J. Baylor, "Parallel Access to Files in the Vesta File System," *Supercomputing '93*, pp. 472-481, November 1993.

[14] Crockett, T.W., "File Concepts for Parallel I/O," *Proceedings of Supercomputing '89*, pp. 574-579, 1989.

[15] Kotz, D., "Multiprocessor File System Interfaces," *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pp. 194-201, 1993.

[16] ParaSoft Corporation, Express C Reference Guide, 1990.

[17] Krieger, O., and Stumm, M., "HFS: A Flexible File System for Large-Scale Multiprocessors," *Proceedings of the 1993 DAGS/PC Symposium*, pp. 6-14, June 1993.

[18] Kendal Square Research, KSR1 Technology Background, January, 1992.

[19] Dibble, P.C., *A Parallel Interleaved File System*, PhD thesis, University of Rochester, March 1990.

[20] Miller, E. L. and Katz, R. H., "RAMA: A File System for Massively-Parallel Computers," *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems*, pp. 163-168, April 1993.

[21] del Rosario, J.M., and Choudary, A., "High Performance I/O for Parallel Computers: Problems and Prospects," *Computer*, vol. 27, no. 3, pp. 59-68, March 1994.

[22] Miller, E. L. and Katz, R. H., "Input/Output Behavior of Supercomputer Applications," *Proceedings of Supercomputing '91*, pp. 567-576, November 1991.

[23] Pasquale, B.K., and Polyzos, G.C., "A Static Analysis of I/O Characteristics of Scientific Applications in a Production Workload," *Proceedings of Supercomputing '93*, pp. 388-397, November 1993.

[24] Pasquale, B.K., and Polyzos, G.C., "Dynamic I/O Characterization of I/O Intensive Scientific Applications," *Proceedings of Supercomputing '94*, pp. 660-669, November 1994.

[25] Reddy, A.L.N., and Banerjee, P., "A Study of I/O Behavior of Perfect Benchmarks on a Multiprocessor," *Proceedings of 17th International Symposium on Computer Architecture*, pp. 312-321, 1990.

[26] Kotz, D. and Nieuwejaar, N., "Dynamic File-Access Characteristics of a Production Parallel Scientific Workload," *Supercomputing '94*, pp. 640-649, November 1994.

[27] Stunkel, C.B., D.G. Shea, D.G. Grice, P.H. Hochschild, and M. Tsao, "The SP1 High-Performance Switch," *Proc. of the Scalable High Performance Computing Conference*, pp. 150-157, May 1994.

[28] Kalns, E., Video on Demand Using Vesta, IBM T. J. Watson Research Center, Yorktown Heights, NY: IBM, September, 1994.

[29] Wu, C.E., Liu, Y.-H., Benveniste, C., Chen, L., and Chiang, W.H., "Trace-Based Analysis and Tuning for Distributed Parallel Applications," *Proc. of the 1994 International Conference on Parallel and Distributed Systems*, pp. 716-723, December 1994.

[30] Wu, C.E., Liu, Y.-H., and Hsu, Y., "Timestamp Consistency and Trace-Driven Analysis for Distributed Parallel Systems," *Proc. of the 9th International Parallel Processing Symposium*, p. to appear, April 1995.

[31] Baker, M.G., Hartman, J.H., Kupfer, M.D., Shirriff, K.W., and Ousterhout, J.K., "Measurements of a Distributed File System," *Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 198-212, 1991.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598