Performance Evaluation of the PVFS2 Architecture

Julian M. Kunkel, Thomas Ludwig
Ruprecht-Karls-Universität Heidelberg, Germany
Institute of Computer Science
Research Group Parallel and Distributed Systems
julian.kunkel@gmx.de, t.ludwig@computer.org

Abstract

As the complexity of parallel file systems' software stacks increases it gets harder to reveal the reasons for performance bottlenecks in these software layers. This paper introduces a method which eliminates the influence of the physical storage on performance analysis in order to find these bottlenecks. Also, the influence of the hardware components on the performance is modeled to estimate the maximum achievable performance of a parallel file system. The paper focusses on the Parallel Virtual File System 2 (PVFS2) and shows results for the functionality file creation, small contiguous I/O requests and large contiguous I/O requests.

1. Introduction

The trend to high performance computing leads to supercomputers consisting of a high number of nodes. While the aggregated computational performance rises it is difficult for the I/O subsystem to keep pace. Parallel file systems like e.g. PVFS2 [**?**, **?**], Lustre [**?**, **?**], and GPFS [**?**] are designed to provide I/O performance that scales well with the number of nodes [**?**]. Physical I/O devices of an arbitrary number of server nodes can be combined by a parallel file system into one logical file system to increase its size and performance. In the best case a parallel file system provides the aggregated performance of all I/O devices to other client applications at a high abstraction level.

However, the achievable aggregated performance of a parallel file system depends on various factors. Considering a single server the capabilities of CPU, network and I/O devices limit the contribution to the file system's overall performance. On the other hand the access pattern of an application and the distribution of data and metadata across the servers defines the level of parallel servers accesses. Moreover the file system itself is a complex parallel program which has its own bottlenecks. Due to these issues we usually see only a certain percentage of the expected throughput as sustained performance.

In order to improve the implementation of a parallel file system developers determine the performance with I/O benchmarks. However, due to the interplay of the components it is hard to identify the reasons for unsatisfactory performance. It might be induced, for example, by the behavior of the servers' I/O subsystem, by the network or by slow CPUs. Also, as a result of the data distribution the requests might utilize only a few servers while others are idle.

This paper introduces a systematic approach for performance analysis of a parallel file system's architecture and shows results for various request types. The idea is to replace the parallel file system's methods accessing the physical I/O system with an efficient stub pretending to be real physical storage. This stub represents the necessary data of the file system in the servers' memory and never triggers real I/O operations. Thus, benchmarking of such a modified file system is not influenced by the slow underlying I/O subsystem.

This paper is structured as follows: At first an overview about the state-of-the-art in performance analysis and related work is given. Then, PVFS2 is introduced in brief. Some simple considerations help to estimate the performance of a parallel file system in section 4. At next, details of our benchmarking concept are described. Then, the evaluation environment and test programs are introduced. A detailed summary of practical results using this methods with PVFS2 is given in section 6. These results are discussed in the last section.

2. State-of-the-Art and Related Work

There are several tools which help to analyze the performance of parallel applications. They can be classified into on-line and off-line tools. Every tool requires a modification of the source code or binary to integrate code for collecting measures of interest. Often, this so-called instrumentation can be done automatically for example by linking the program against a special library.

On-line tools like Paradyn [?] collect and display the measured data at run-time and are even able to control the overhead imposed by the tool. Other tools even allow a direct manipulation of the program. However, these tools are parallel programs by themselves, thus complex, and due to their resource requirements they influence the run-time behavior of the application. Off-line tools are not so flexible, but easier to implement. They either collect statistical data during a time period (profiling) or information about events during run-time (tracing). This information gets stored and can be analyzed after program completion. For example the MPE [?] tracing library saves events triggered by MPI function calls in trace files, which can then be analyzed by the jumpshot viewer of MPICH2 [?]. There are also a couple of commercial tools like e.g. the Intel Trace Analyzer (the former Vampire [?]).

Various I/O benchmarks are available for regular local file systems, e.g. Bonnie [?], IOzone [?] and PostMark [?].

As parallel and network file systems can be accessed with a vast amount of access patterns and mostly are optimized for specific tasks, there is currently no common benchmark available that covers a representative range of possible access pattern [?]. In most cases either code of a scientific application or simply a serial benchmark is used to determine the performance. A well known and often applied benchmark is b_eff_io [?]. Its purpose is to determine the file system performance with different characteristic access patterns. However, the interpretation of measured performance values in order to detect bottlenecks in the system is complicated due to the complex interplay of the components.

Replacing the physical I/O subsystem with a simple stub, i.e. with dummy functionality, eases the task to find bottlenecks in the implementation of a parallel file system. This idea is derived from software testing where the correctness of code segments is verified with stubs only providing the code necessary for testing. One might argue that the storage space of a parallel file system could be placed on a in-memory file system, for example on Linux tmpfs, in order to gain similar results. That is partly true, though in-memory file systems are limited to the size of the memory, so benchmarking of a large amount of data is not possible. However, an in-memory file system is appropriate for small datasets and analysis of metadata performance. Finally, the efficiency of the storage layer itself can be determined by comparing tmpfs results with the results of an efficient stub.

3. Overview of PVFS2

The parallel virtual file system PVFS2 [?] is a redesign of the first version of the file system aiming at better modularity, flexibility, and a tight MPI-IO integration.

A PVFS2 server holds exactly one so-called storage-space, which may contain a part of several logical file systems. In terms of PVFS2 a logical file system is called collection. According to the type of storage provided for a file system, servers can be categorized into data servers and metadata servers. A distribution function controls the way data is spread over the available data servers. In most cases it gets striped over multiple nodes using local files of a Unix file system. Metadata servers store object attributes. This is all the information about files in the Unix sense, i.e. object type, ownership and permissions. Additional information like the directory hierarchy, is stored on metadata servers, too. A compute node can be configured as either a metadata server, a data server, or both at once.

PVFS2 has the layered architecture illustrated in figure 1. Interfaces between layers use a non-blocking semantics. The user-level interface provides a high abstraction to a PVFS2 file system. Currently, there are integrations with MPI-IO and the kernel VFS available. The system interface API provides functions for the direct manipulation of file system objects and hides internal details from the user. Invoking a request starts a dedicated statemachine processing the operation in small steps. Statemachines break complex requests into several states each representing an atomic operation. Clients and servers can interlock the execution of these operations to obtain a time-shared processing of different requests. A specific execution order is chosen to ensure that a client crash has no impact on the metadata consistency.

This layer also incorporates two caches, which store informations about the directory hierarchy and object attributes to avoid repeated server requests. Unlike NFS and other network file systems PVFS2 does not cache I/O operations on the client side. The job layer consolidates the lower layers into one interface and maintains thread functions for these layers. Data of a larger I/O operation is directly transferred between two endpoints by Flow. An endpoint is one

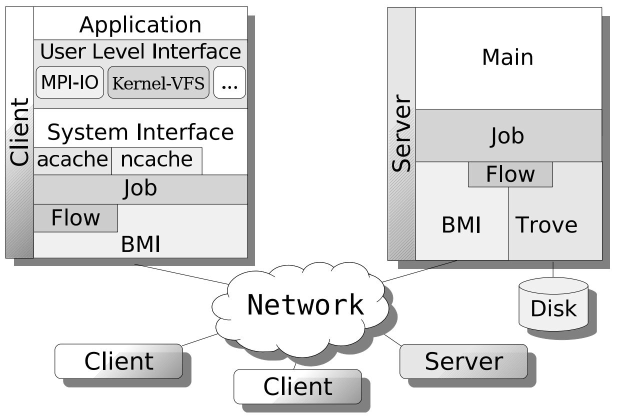

Figure 1. PVFS2 software architecture

out of memory, network, or persistency layer. Flow takes care of the data transmission itself once the endpoints are specified. The Buffered Message Interface (BMI) provides a network independent interface. Clients communicate with the servers by using the request protocol, which defines the message structure and contents. BMI can use different communication methods, currently TCP, Myricom's GM, and Infiniband. On the server side a main process decodes incoming requests and starts a new instance of the request's dedicated statemachine. Trove [**?**] is the persistency layer providing methods for manipulation of key/value pairs (used for metadata) and data streams.

BMI, Flow and Trove are modular and the actual implementation can be chosen by the user. Currently, there is only one Trove module available, database plus file (DBPF) [**?**], which stores metadata in Berkeley databases and data in Unix files. We will replace this very same component in order to conduct our benchmarking approach.

4. Performance Limitations

In this section some simple considerations lead to estimated upper bounds for the aggregated throughput of small and large I/O requests. Therefore, the influence of the I/O subsystem, network, and CPU are discussed. As these components limit strongly the performance of a parallel file system, it is important to understand their influence. Communication between different machines is limited by the network performance. The following network characteristics are important: *Latency* is the time between the sending of a message and its arrival at the receiver side. Usually we evaluate this time with empty message body. *Bandwidth* is the number of bits which can be transferred in a specific time. Because protocols like TCP have some overhead and control algorithms, the *throughput* is smaller than the bandwidth. Latency and bandwidth depend on the used network technology and topology. Latency is also influenced by the distance.

The I/O subsystem normally consists of a set of hard disks. Important characteristics of a hard disk are access time and transfer rate. *Access time* is the time needed to seek a data block. It depends on the current position of the disk's heads and the target block. In average the access time is a few milliseconds. Once the heads are placed, subsequent blocks of the cylinder can be read or modified very fast, which results in a higher *transfer rate*. In order to improve performance, a disk typically prefetches and caches blocks. Additionally, the operating system buffers a fair amount of an I/O operation depending on the amount of free memory. A write operation can be buffered efficiently, so it can complete before data is actually written to disk. A read operation can completely omit any I/O operation if the data is in memory. Otherwise, the operation has to wait for the data to be available.

The server's CPU speed and architecture define the time needed to process instructions. PVFS2 mostly uses efficient data structures like hash tables. In comparison to the network latency and storage subsystem's access time, the CPU is the fastest component. Especially if we connect various clients to one server, CPU is not expected to limit throughput. Assume that each request needs the same time for processing. Then the number of requests which can be processed in a time interval is determined by the CPU's capabilities.

As a result performance of small requests greatly depends on the network latency and the disks access time, while large requests are bound by the network throughput and disk's transfer rate. Due to the fact that current disks are slower than the network, measured performance is greatly influenced by the disks' capabilities. Also, it is hard to predict the benefit of the disk caching strategies.

In order to estimate the throughput for large contiguous I/O requests we assume the following facts: There are 5 I/O servers, each equipped with 1 GBit/s Ethernet and a hard disk with an average transfer rate of 40 MByte/s. Also, there is a disjunctive set of clients each equipped with 1 GBit/s Ethernet. With tools like netperf, which are designed to analyze network performance, network throughput can be estimated better. Figure 2 shows the aggregated performance limits of network and I/O subsystem for the servers and a variable number of clients. On the server side the actual performance is the minimum of disk throughput and network throughput (horizontal lines), on the client side the actual performance will be on the right side of the diagonal. Thus the observed performance will be in the segment right of the diagonal and below the lower horizontal line.

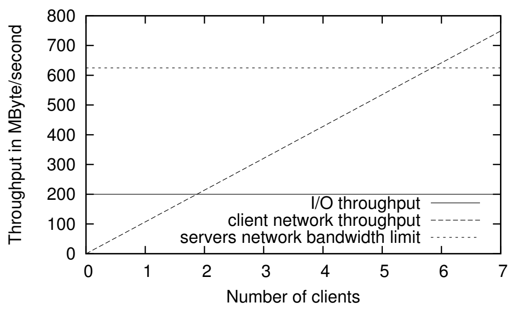

Figure 2. Estimated performance limits for large I/O requests

Further information about internals of a parallel file system help to tighten the upper bound. However, sometimes it is not easy to obtain such information, especially if an inspection of the source-code is not possible and optimizations on different layers interact. In the following we incorporate knowledge about some PVFS2 internals. The lack of a client side caching mechanism for I/O operations entails the cost of at least one network round-trip-time. Small I/O accesses which fit into the initial request or response could be transmitted with one round-trip (the size of data that fits in the packet is about 16 KByte). Bigger accesses require a rendezvous protocol and a handshake which can be transferred directly before the I/O is started for reads but not for writes. Thus the transfer needs an extra round-trip for write operations. This knowledge is incorporated in the estimations for small I/O accesses (figure 3 and 4).

Modifying metadata operations are not cached and may consist of multiple requests, which typically have to be processed in serial order to guarantee a consistent file system. E.g. in order to create a file 4 requests are processed (5 with MPI): one to verify if an object with the same name exists (necessary in MPI_File_open), one to create a new metafile object on the metaserver, one to create a datafile on each dataserver (requests can be processed in parallel), one to write the handles of the datafiles in the metafile and one to update the directory entry. All steps require at least 5 modifying operations which enforce synchronization of the disc.

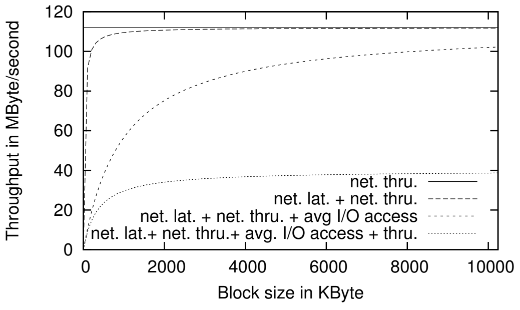

Figure 3. Estimated performance limits for small I/O requests of one client

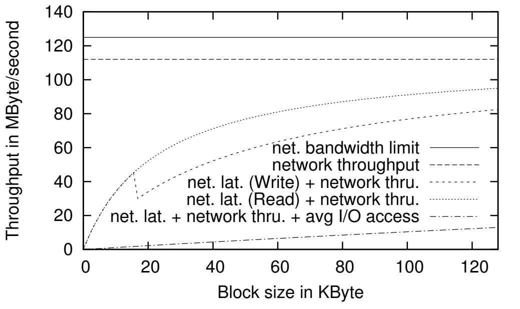

Figure 4. Estimated performance limits for small I/O requests of one client - small blocksizes

5. Evaluation Setup

At first, we design and implement a new Trove module, the Trove Analyzation Stub (TAS). It uses red-black-trees as basic data structures to provide a fast in-memory management of metadata. Metadata is handled correctly to support all kinds of benchmarks. I/O requests are discarded while the size of a file is adapted in the metadata and success of the operation is signaled to other layers immediately. For further technical details refer to [**?**].

In order to compare and evaluate the two different persistency modules TAS and DBPF the simple MPI programs `mpi-io-test` and `mpi-md-test` (shipped with PVFS2) benchmark the performance of several client and server configurations.

`mpi-io-test` first writes a number of blocks of the same size with `MPI_File_write` and then opens the file again and reads the data back with `MPI_File_read`. In each iteration `MPI_Barrier` ensures that all processes are ready before the actual I/O operation is done. The position is then set with `MPI_File_seek`. During the process it measures the time needed for the I/O operations and calculates the bandwidth. Multiple clients access the blocks of the file in a round-robin fashion e.g. process one accesses the first block of the given size, process two the second block and so on.

To benchmark metadata operations the MPI program `mpi-md-test` runs the same number of `MPI_File_open` calls on each client to create a file. The program is adapted to supported individual operations per client.

It is interesting to see the influence of the number of clients and servers and the accessed file size. On the working group's cluster configurations up to five servers were tested. Clients can be distributed on the server nodes, then a maximum I/O throughput is possible due to bypassing the network for access of local disk. Also, disjoint machines can be used for client and servers. Normally, induced by timing and hardware effects performance varies slightly between different test runs. Therefore throughput for the same input parameters is measured three times. Also, to guarantee an identical environment for each run for one test, the servers are restarted and the storage space is recreated. In the evaluation benchmark results are compared with the estimated upper bounds to allow a qualitative analysis.

One testbed is the working groups cluster, which consist of 10 nodes each equipped with two Intel Xeon 2000 Mhz processors, 1 GByte main memory, Gigabit Ethernet which is interconnected via copper cable using a star topology, and an IDE hard disk which has a throughput of about 40 MByte/sec. DBPF uses an ext3 partition mounted with the options `commit=60` and `data=writeback` to store persistent data.

To provide new result and refer to some former results the version of PVFS2 used for the benchmarking differs and are either from December 2005 (results from the Bachelor's Thesis [?]), July or October 2006. If the version used for testing is not the recent october version this is specified in the diagram titles. In the meantime the working groups cluster has been reinstalled as well and some TCP options were tuned.

6. Results with PVFS2

In this section first we present the results for metadata operations, then for small contiguous I/O access of a 100 MByte file and for large contiguous I/O access of big files.

Figure 5 shows the achieved number of create operations per second benchmarked with the adapted `mpi-md-test` on different configurations of one metadata server. Note that the clients are distributed over nine nodes in round-robin fashion. For comparisons the in-memory file system tmpfs is shown and a configuration setup where the server is not forced to sync modifying operations to disk. An observation is that the performance of DBPF without syncing is close to DPBF on top of tmpfs which is expected. Earlier versions of PVFS2 organized metadata in a different disk layout which resulted in an inefficent non-syncing version [?]. The diagram shows that it is important to develop clever mechanisms to guarantee consistency without the necessity to force synchronization on each modifying operation. In the meantime the developers incorporated a mechanism into PVFS2 to coalesce multiple metadata requests into one synchronization, however this is out of the scope of this paper.

Performance of I/O accesses for block sizes between 1 KByte up to 10 MByte is measured for one and five dataservers, the aggregated size of the file is always 100 MByte which is expected to be cached well. In figure 6 the throughput of TAS is compared with the estimated bounds. The estimation is a asymptote to the performance of TAS. Diagrams 7 and 8 highlight the fact that TAS performance is an upper bound for each implementation, even if the data fits into memory. A reduction of the write performance due to the change to the rendezvous protocol can be seen at 16 KByte for TAS but not for DBPF. Additional testing with for example DBPF on tmpfs could help to identify inefficiencies. Also the throughput of DBPF increases close to the network bandwidth limit for larger block sizes. During the recent test for small block sizes it turned out that performance of TAS and the immediately completion was slower than DBPF's queuing mechanism (only about 50%). Testing indicates an inefficiency in the layers, which can be bypassed by using only one parallel data flow for TAS. Thus these results are shown in these diagrams. The modification of the number of flows however does not increase the performance of DBPF, thus it uses the default of 8.

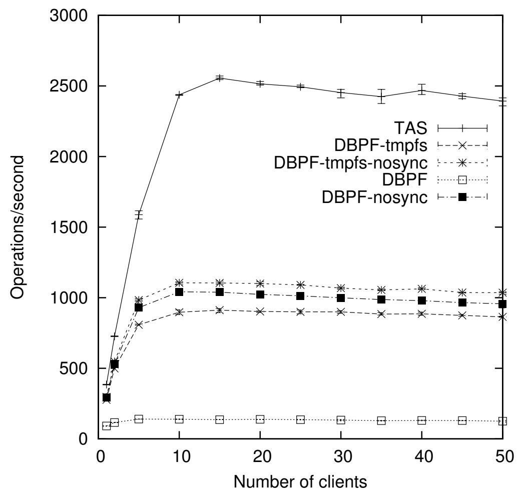

Figure 5. Results for MPI file creation test (PVFS2: July)

At first a comparison of the throughput for one server and one client with the performance of 5 clients and 5 servers (presented in figures 9 and 10) seem to indicate that the throughput does not scale with the numbers of servers. One server archives the throughput of 44 MByte/second for read and 32 MByte/second for write while five servers achieve only a throughput of 128 MByte/second for read and 118 MByte/second for write requests. This is only roughly three times faster than for one client and server. To understand this behavior it is necessary to look at the striping mechanism. By default PVFS2 stripes data in 64 KByte chunks over the servers.

Depending on the access pattern the load might be imbalanced or at worst some servers might be idle at a time

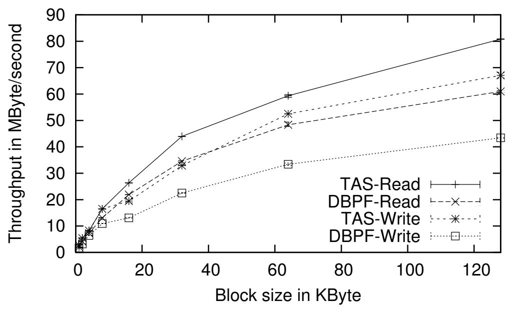

Figure 6. I/O throughput for one client accessing a variable block size and estimated bounds

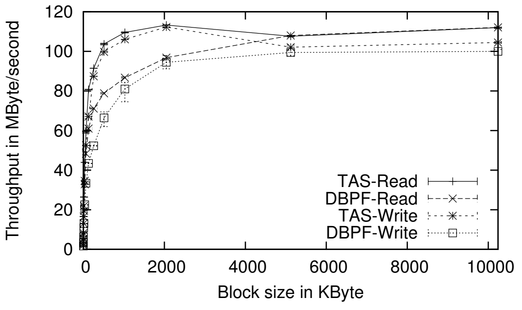

Figure 7. I/O throughput for one client accessing a variable block size

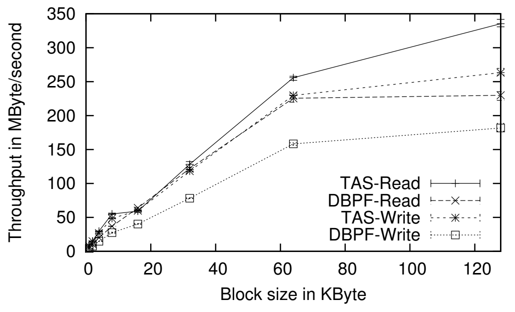

Figure 10. I/O throughput (5 clients, 5 servers)

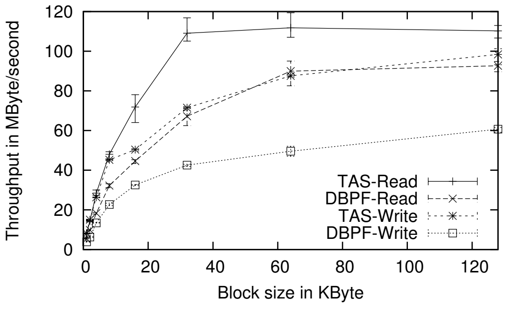

Figure 8. I/O throughput (5 clients, 1 server)

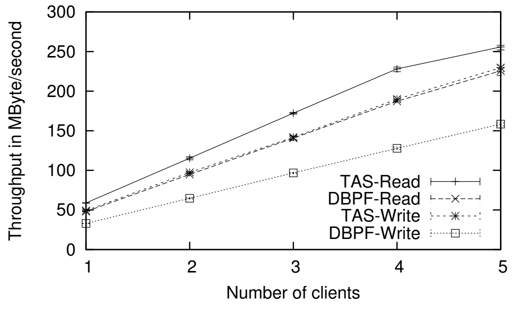

Figure 11. I/O throughput for a blocksize of 64 KByte (5 servers)

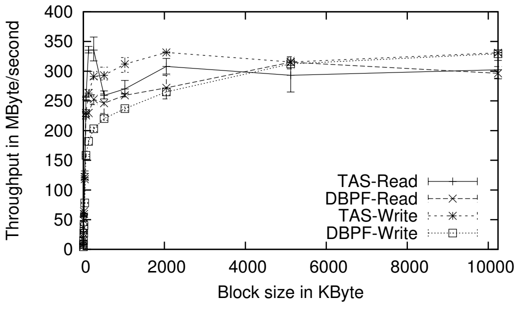

Figure 9. I/O throughput (5 clients, 5 servers)

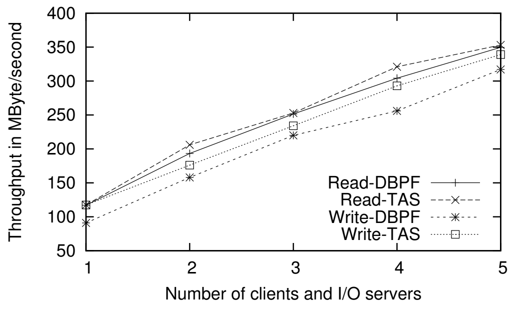

Figure 12. I/O throughput for a blocksize of 10 MByte (5 servers)

reducing the maximum concurrency. This is the case for mpi-io-test running with smaller block sizes than the stripe size. For a block size of 32 KByte only three servers are hit during I/O, two servers have to access two times 32 KByte and another one has to access one block.

Freezing the block size to a multiple of 64 KByte and increasing the number of clients the performance should scale linearly which is shown in figure 11. Unfortunately, for big block sizes like 10 MByte the gradient is suboptimal (figure 12).

To measure performance of large contiguous I/O requests during each iteration the MPI call of mpi-io-test transfers 10 MByte of data until the desired file size is reached. Figure 13 shows older results for a single client accessing only one file which has a varying file size.

Looking at the diagram we can see that TAS read performance sticks behind the write performance. This indicates a bottleneck in the architecture: internally read and write operations are treated similarly, however, the flow protocol has a different implementation. The write throughput (116 MByte) slightly surpasses netperf's results

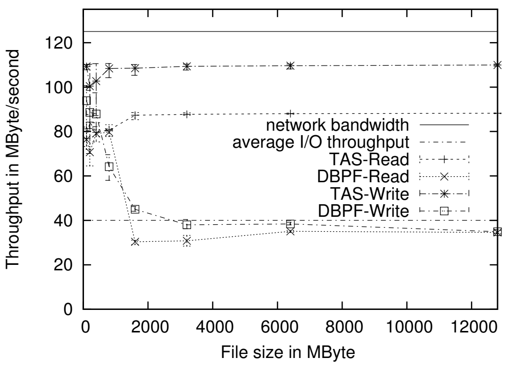

Figure 13. I/O throughput for 1 client accessing a file with varying size (PVFS2: Dec.05)

(112 MByte) and is even close to the network bandwidth (125 MByte). Our concept shows that certain problems in the implementation are hidden by the fact that the slow disk access dominates overall throughput. In the meantime this specific bottleneck is tracked down and eliminated (compare figure 15).

Caching effects of the disk and linux kernel can be seen for the througput of DBPF. The MPI program first writes the whole data then waits until all clients are finished and rereads the data. This behavior explains that read requests are handled efficiently up to 800 MByte, which is almost the available main-memory. Write requests might benefit up to big files because they need not wait for the disk to finish an operation and instead cache the data in memory and keep the disk busy.

In figure 14 the performance of a variable number of clients writing together 12.800 MByte to a single file is given. While the client number is increased TAS read and write performance converge to a value close to the servers network bandwidth. This suggests that PVFS2 is well designed for this use case. However, the real read performance drops to half of its value when benchmarking one client and two clients. Probably this loss can be traced back to the Linux kernel's asynchronous I/O implementation (AIO) which is used by DBPF. Also, while the client number increases the I/O performance decreases.

This might be induced by the local file system because a disks performance decreases for multiple streams due to necessary head movements. With the new PVFS2 version and reinstalled testing environment the throuput of TAS sticks to 116 MByte for all number of clients and servers. Also the performance does not drop for DBPF.

Figure 15 shows the measured throughput for the current PVFS2 with a variable number of clients and servers accessing a file which has a total size of 12.800 MByte. While TAS throughput grows linearly performance for read operations does not for DBPF. This points out that network is not the bottleneck for DBPF. The issue is unresolved and has to be investigated in the future.

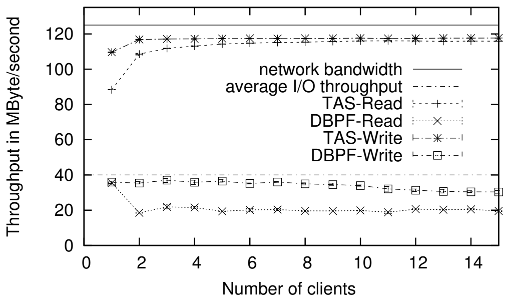

Figure 14. I/O throughput for a variable number of clients acessing a 12800 MByte file (PVFS2: Dec.05)

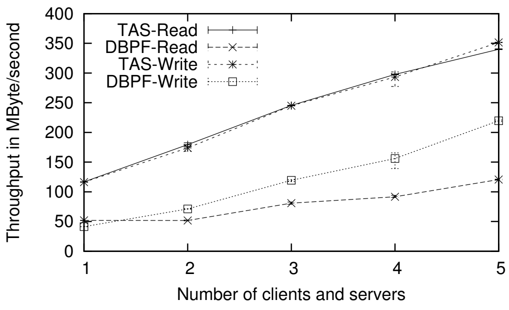

Figure 15. I/O throughput for a variable number of clients acessing a 12800 MByte file

On Argonne National Laboratory's Chiba City cluster tests with a higher number of clients and servers have been run. Chiba's machines consist of dual 500 MHz PIII with 512 MByte and are interconnected with Myrinet 2000 cards. However, the effective network throughput between two nodes was only about 90 MByte. Testing revealed that the network interconnect has some issues, sometimes even the same number of packages have to be transmitted twice resulting in bad performance. In this test the number of clients and servers is increased in the same proportion, each client accesses a total of 150 MByte of data which fits into the servers' memory. In figure 16 the best of three runs is shown. It can be seen that the expected performances scale with the number of clients and servers almost linearly. Note that the used version of PVFS2 is older.

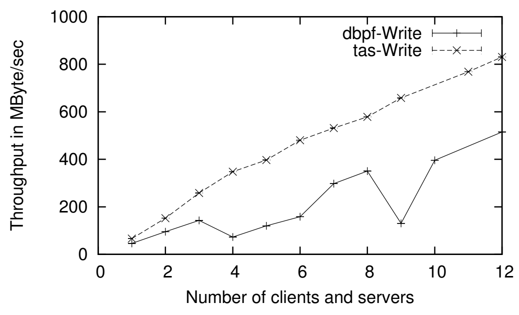

Figure 16. I/O throughput for a variable number of clients each accessing 150 MByte of data (PVFS2: July)

7. Conclusions and Future Work

There is no benchmark available which is suitable for an analysis of a parallel file system that reveals internal performance bottlenecks. At first we want to have reference values for performance measures that represent best case values. Any improvement of internal software layers will be evaluated with respect to these best case values. The TAS module, which is a dummy persistent storage module for PFVS2, acts as a reference for throughput achievable from the other Trove modules, which do real physical I/O. With the help of such an upper bound the real costs of a module's I/O strategy are pointed out. Also, the effect of changes in the other layers can be observed better. It seems to be important to do performance regression testing to keep track of the performance changes. Further performance details that refer to the December 05 PVFS2 version can be found in [**?**] and comparision with the new results show the improvements of the PVFS2 project in this period.

In our approach the real persistency layer is replaced by a dummy that resembles a correctly working disk I/O layer. By running benchmark programs we could find several bottlenecks in the PVFS2 implementation, for data I/O as well as for metadata management. As a project in parallel to this we design and implement a Jumpshot based tool environment that will give a better insight into server performance details.

References

[1] T. Bray. Homepage of Bonnie. http://www.textuality.com/bonnie/.

[2] I. Cluster File Systems. Lustre: scalable, robust, highly-available cluster file system. Online-document http://www.lustre.org/, 2006.

[3] J. Katcher. Postmark: a new file system benchmark. Technical report TR3022. Network Appliance, Oct. 1997.

[4] J. M. Kunkel. Performance Analysis of the PVFS2 Persistency Layer. Bachelor's thesis, Ruprecht-Karls-Universität Heidelberg, Germany, Institute of Computer Science, Research Group Parallel and Distributed Systems, Feb. 2006.

[5] A. N. Laboratory. http://www.pvfs.org/. Online-document http://www.pvfs.org/, 2006.

[6] R. Latham, N. Miller, R. Ross, and P. Carns. A next-generation parallel file system for linux clusters. *Linux World Magazin*, 1, 2004.

[7] J. Layton. Cluster Monkey: Benchmarking Parallel File Systems. Online-document http://www.clustermonkey.net/content/view/62/32/, 2003.

[8] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, 1995.

[9] MPICH2 Team. Homepage of mpich2. http://www-unix.mcs.anl.gov/mpi/mpich/.

[10] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[11] W. D. Norcott. Homepage of IOzone. http://www.iozone.org/.

[12] PVFS2 Development Team. Trove Database + Files (DBPF) Implementation. PVFS2 Documentation included in the source code package, 2005.

[13] PVFS2 Development Team. Trove: The PVFS2 Storage Interface. PVFS2 Documentation included in the source code package, 2005.

[14] R. Rabenseifner and A. E. Koniges. Effective communication and file-I/O bandwidth benchmarks. *Lecture Notes in Computer Science*, 2131:24+, 2001.

[15] A. Saify, Kochhar, G., J. Hsieh, and O. Celebioglu. Enhancing high-performance computing clusters with parallel file systems. *Dell Power Solutions*, May 2005.

[16] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the Usenix FAST 2002 Conference*, pages 231–244, 2002.

[17] P. Schwan. Lustre: Building a file system for 1000-node clusters, 2003.

[18] C.-F. E. Wu, A. Bolmarcich, M. Snir, D. Wootton, F. Parpia, A. Chan, E. L. Lusk, and W. Gropp. From trace generation to visualization: A performance framework for distributed parallel systems. In *Supercomputing*, 2000.