vPFS: Performance Virtualization of Parallel Storage Systems

Yiqi Xu*, Dulcardo Arteaga*, Ming Zhao*, Yonggang Liu†, Renato Figueiredo†, Seetharami Seelam¶
* Florida International University, {yxu006, darte003, ming}@cs.fiu.edu
† University of Florida, {yonggang, renato}@acis.ufl.edu
¶ IBM T.J. Watson Research Center, sseelam@us.ibm.com

ABSTRACT

There remains an inadequacy in parallel file system performance management approaches to support the allocation of shared storage resources on a per-application basis. We propose vPFS, which provide existing parallel file systems the complementary abilities to differentiate I/O requests from different applications and meet per-application Quality of Service (QoS) requirements. vPFS employs user-level parallel file system proxies to interpose requests between native parallel file system clients and servers to schedule parallel I/Os from different applications according to configurable bandwidth management policies. The architecture of vPFS is designed to be generic enough to support various scheduling algorithms and apply to different parallel file systems. Specifically, a prototype that virtualizes PVFS2 with enhanced SFQ-based schedulers is implemented and evaluated in this paper. Results obtained from typical HPC benchmarks show that the vPFS approach can achieve high utilization and good bandwidth isolation.

1. INTRODUCTION

The I/O bandwidth that an application gets from the shared storage system in high-performance computing (HPC) systems is critical to the application's Quality of Service (QoS). In a large HPC system, competing applications may have distinct I/O characteristics and demands that result in significant performance interference. However, existing parallel storage systems see only generic I/O requests arriving from the compute nodes, and they are incapable of satisfying the applications' different I/O needs.

This paper presents a new approach — vPFS — which addresses these challenges through the virtualization of existing parallel file systems, in order to achieve application-QoS-driven storage resource management. With vPFS, various I/O scheduling algorithms can be realized at the proxy-based virtualization layer for different storage management objectives. Specifically, this paper considers Start-Time Fair Queueing (SFQ) [2] for proportional sharing of storage bandwidth sharing guarantee. This paper further proposes and evaluates synchronization improvements to SFQ motivated by the unique characteristics (parallel striped I/Os) and requirements (asymmetry of file layouts) of a parallel storage system.

The proposed approach is implemented upon PVFS2 [1] and evaluated using typical parallel computing and I/O benchmarks (IOR [5], NPB BTIO [6]). The results show that this approach achieves good proportional bandwidth sharing (at least 95% of the target sharing ratio) for competing applications with diverse I/O patterns.

2. APPROACH

To solve the architecture limitation on parallel storage and realize schedulers for it, we designed the architecture of the vPFS as shown in Figure 1. It is built upon the typical HPC system architecture, where applications access their data on a parallel storage system that mainly consists of a parallel file system (PFS) and its associated storage networks and devices. A virtual PFS can be dynamically created for an application by spawning a proxy on each involved server which brokers the application's I/Os. This performance virtualization of parallel storage in an HPC system is designed to be more knowledgeable about application intricacies and more capable of managing the QoS compared with the native storage system. First, it is able to recognize different applications' I/Os instead of treating them as the same. Second, it is also able to re-order the I/O requests at the storage utility side for bandwidth management based on the QoS specified by each application. Third, different parallel I/O schedulers can be enabled upon the virtualization layer.

To achieve proportional bandwidth sharing, vPFS implements schedulers enhanced upon existing SFQ-based algorithms [3][4] through cooperating proxies. These proxies are responsible of not only scheduling the requests serviced by its local data server but also exchanging the local service information among one another in order to achieve total-service proportional sharing. Two complementary schemes are studied on efficient synchronization of global scheduling information across distributed proxies. In Scheme 1, the proxies can reduce the frequency of synchronization by batching the costs of a number of locally serviced requests in a single synchronization message. Furthermore, the synchronization frequency can be adjusted in each time window based on a specified threshold of serviced throughput. In Scheme 2, the cost of synchronization can be further reduced from utilizing the layout information of the files accessed by the

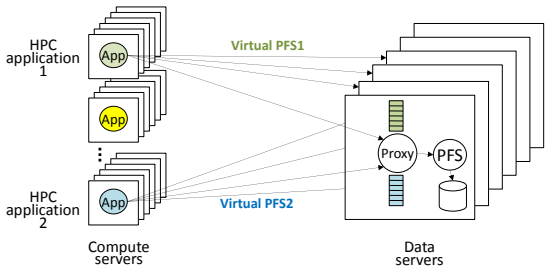

Figure 1. The architecture of vPFS

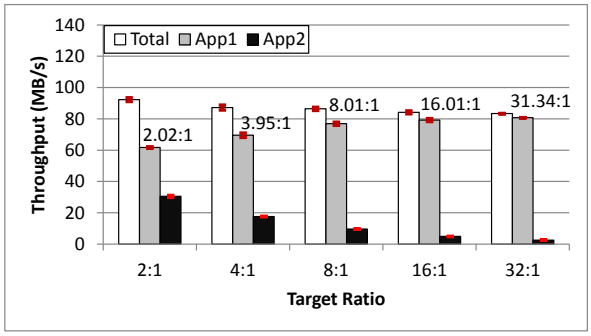

Figure 2. Bandwidth sharing between two sets of IORs

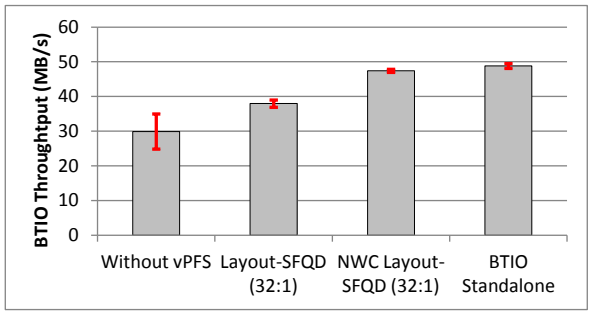

Figure 3. Throughput of BTIO when running with IOR

applications. The obtained layout information will be used to re-construct the original application level parallel I/O request representing the accurate global service amount for use by global proportional sharing.

3. EVALUATION

The PVFS2-based vPFS prototype was implemented and evaluated on a testbed consisting of two clusters, one as compute nodes and the other as I/O nodes running PVFS2 (version 2.8.2) servers. The compute cluster has eight nodes each with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and one 500GB 7.2K RPM SAS disk. The server cluster has eight nodes each with two six-core 2.4GHz Intel Xeon CPUs, 24GB of RAM, and one 500GB 7.2K SAS disk. Both clusters are connected to the same Gigabit Ethernet switch. All the nodes run the Debian 4.3.5-4 Linux with the 2.6.32-5-amd64 kernel and use EXT3 as the local file system. IOR (2.10.3) [5] is used to generate synthetic sequential and random I/Os. BTIO from the NAS Parallel Benchmark (NPB) suite (MPI version 3.3.1) [6] is used to represent a typical scientific application with interleaved computation and I/O phases.

In Figure 2, each application occupies 4 separate compute nodes exclusively and uses 32 processes on each node. The layout of App1 is simple stripe on 4 I/O nodes and the layout of App2 is simple stripe on all the 8 I/O nodes. The applied algorithm is layout-based SFQ. App1 issues random reads and writes while App2 issues sequential writes. The figure shows that the achieved throughput isolation (on the bars) between two applications is within the 95% range of various desired goals. Figure 3 compares the effectiveness in restoring the throughput to the BTIO standalone mode when using layout-based SFQ scheduler and its non-work-conserving variation. X axis shows the scheduler and its parameters used if both BTIO and IOR are in the system. BTIO workload is collective and large, faced with the same number of competing IORs (64 processes with sequential writes). In contrast, BTIO has a bursty access pattern and has limited concurrency. The layout-based SFQ made an improvement to BTIO by 27% but does not restore its throughput completely because of intensive I/O interference from IOR with aggressive I/O issue rates. Non-work-conserving layout-SFQ isolated the performance much more effectively under intensive contention and improved the throughput by 58%.

4. CONCLUSION AND FUTURE WORK

This paper presents a new approach, vPFS, to provide a performance virtualization framework and proportional sharing schedulers built upon it for parallel storage management in HPC systems. Experiment results show that the vPFS approach is feasible and it can achieve nearly perfect total-service proportional bandwidth sharing for two competing parallel applications with diverse I/O patterns. In the future we will evaluate vPFS with larger scale setups considering more servers and applications. We also plan to realize a latency-driven scheduler to provide response time bound in parallel applications among diverse applications via feedback controlled, self-adaptive methods.

5. REFERENCES

[1] PVFS2. URL: http://www.pvfs.org/pvfs2/.

[2] P. Goyal, H. M. Vin, and H. Cheng, “Start Time Fair Queueing: A Scheduling Algorithm For Integrated Services Packet Switching Networks”, IEEE/ACM Trans. Networking, vol. 5, no. 5, 1997.

[3] W. Jin, J. S. Chase, and J. Kaur, “Interposed Proportional Sharing For A Storage Service Utility”, SIGMETRICS, 2004.

[4] Yin Wang and Arif Merchant, “Proportional Share Scheduling for Distributed Storage Systems”, FAST, 2007.

[5] IOR HPC Benchmark, http://sourceforge.net/projects/ior-sio/.

[6] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Resources/Software/npb.html.