SAND2009-2796C

On Scaling I/O for Commodity Clusters

Sandia National Laboratories
Benjamin Allan, Helen Chen, Scott Cranford, Ron Minnich, Don Rudish, and Lee Ward

Abstract

We benchmarked and improved the single-process streaming write efficiency of the Linux NFS client. Deeper analyses of the benchmarks and of various I/O short-circuit schemes establish upper bounds on the performance of the NFS client even with an infinitely fast network. The complex interactions of the Linux Virtual File System and Virtual Memory Management apparently impose a limit on further improvement which is substantially less that the theoretical streaming bandwidth of the fast interconnects.

1. Overview of the Problem and Idea

Our goal is to enable a high performance parallel file system that is an integral part of the mainline Linux kernel [1]. Our initial benchmarks indicate that only a small fraction of the available bandwidth on high-speed transports (InfiniBand [2], 10 Gbps iWARP [3], and TCP) is utilized when streaming writes are made to a Linux NFS volume. While specialized distributed parallel filesystems have been developed outside of the Linux main stream, they have not proven to be satisfactory from the standpoint of community support, reliability, and performance. The Linux community is improving the kernel-standard NFS file system, but the work is not aimed at HPC needs [4]. In order to address the HPC community, a new type of NFS, called Parallel NFS, or pNFS [5], has been proposed. One obstacle in scaling pNFS using NFS RDMA for Storage I/O is the performance bottleneck in NFS streaming-writes, which stems from Linux's NFS implementation and its interaction with the Linux virtual file system (VFS) [6] and virtual memory management system (VMM) [7]. Write throughput is strongly affected by the lack of concurrency between application I/O and the NFS implementation's flushing of cached data based on our initial experience [8].

We identified two strategies to optimize streaming writes on the client side. The first strategy is to re-implement the Linux NFS client to be procedure-based, multi-threaded and asynchronous. This approach is unacceptable by the Linux community at large as it would be a giant replacement of a kernel subsystem which most view as robust and fully functional, and so we decided not to pursue further. The second strategy is to improve the behavior of the Linux VFS and VMM as they interact with NFS to increase overlapping of application and network I/O.

2. Benchmark Profiles and Rate Results

2.1 Research Tools

We generated performance debugging data with instrumented source code of the NFS server, the NFS client, and the Linux file and cache subsystems while running performance benchmarks. We also collected large granularity performance profile data with a tool (collectl) [9] that queries the /proc filesystem [10] on un-instrumented kernels.

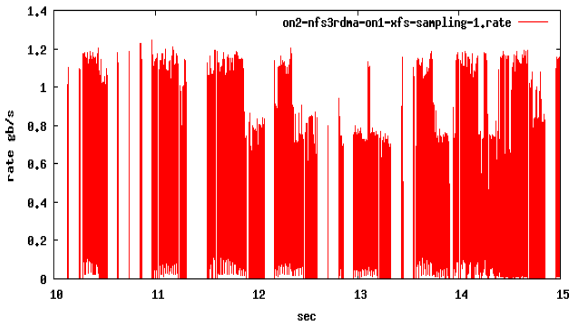


Figure 1 I/O Profile of NFS Streaming Write

We present results collected with iozone [11] and collectl in the following subsections. The test setup consists of a single NFS-RDMA server with 8GB of RAM, 2 AMD Opteron 248 Processors, a DDR InfiniHost III HCA, with a TYAN S2895 motherboard. Our client node has a similar setup with the exception of having only 4 GB of RAM. They are connected through a 24 port SilverStorm IB switch at Double Data Rate.

2.2. The Linux 2.6.16 Kernel

Our initial benchmark was conducted under the Linux 2.6.16 kernel with the earliest stable implementation of NFS RDMA. We plotted the traffic profile of a streaming-write test over the IB RDMA transport in Figure 1 and found that NFS writes were periodically throttled. We used collectl to gather system statistics during a streaming-write test at 10 Hertz in an attempt to identify and understand the bottlenecks in the Linux kernel. We collated these statistics in Figure 2 to study the dynamic between the CPU, Memory-cache, NFS-write, and the InfiniBand Network subsystems. These plots clearly reflect the stop-and-go I/O profile demonstrated in Figure 1; when the usage of memory cache reaches the 34% threshold, Linux's Virtual Memory Management throttles the user application and starts to flush cached data through NFS writes, resulting in visible increases in network traffic. With sufficient cache memory reclaimed, the network

traffic is stopped, and the CPU and memory cache usage starts to rise, indicating the start of the next iteration of application I/O; we see no overlapping between the application and the network I/O.

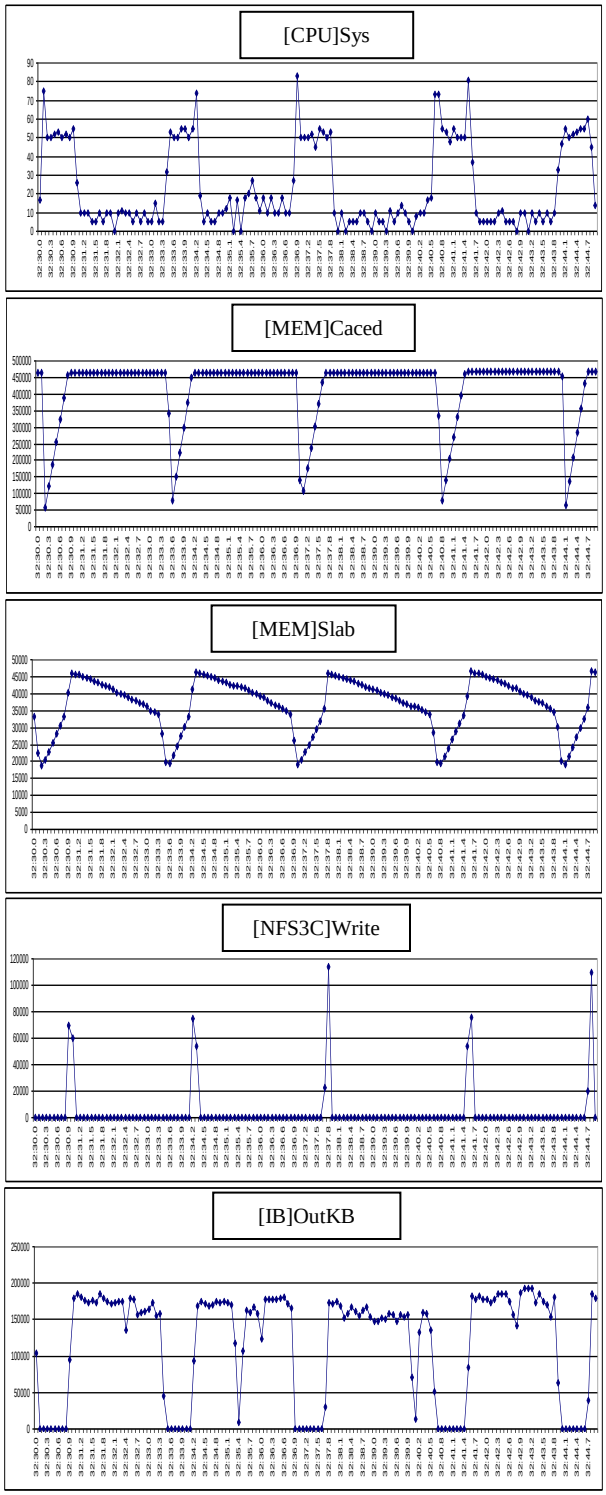


Figure 2 Collectl Statistics of The 2.6.16 kernel – CPU, Memory, NFS-write, and IB Traffic

2.3. The 2.6.26 Kernel

We upgraded the Linux kernel from 2.6.16 to 2.6.25, and 2.6.26 in order to synch up with bug fixes and new features by kernel developers. We repeated the same streaming-write benchmark and plotted collectl statistics in Figure 3 to compare their traffic profiles. As shown, the original stop-and-go symptom in the 2.6.16 kernel had been fixed in 2.6.25 and 2.6.26; however, the overall network efficiency remained sub-optimal at only 1/10 of the available 16 Gbps bandwidth.

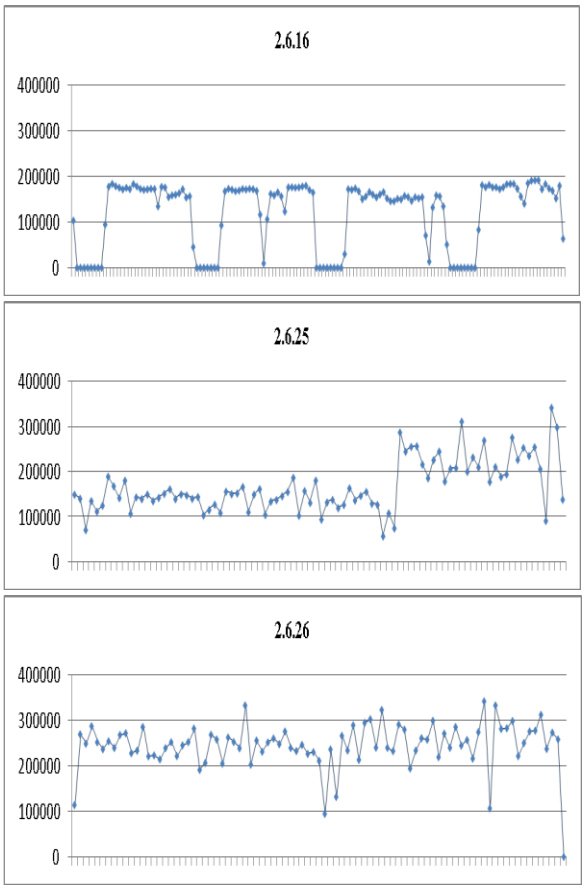


Figure 3, a Snap Shots of Collectl Statics Comparing the Traffic profile of NFS Streaming Write under Linux Kernel 2.6.16, 2.6.25, and 2.6.26

2.3.1. I/O Short Circuit Schemes

Continuing to locate causes for the low performance, we developed two kernel patches to short circuit the data flow as depicted in Figure 4. The first patch short-circuits the server-side NFS-write to bypass the server filesystem and disk I/O.

This patch stubs the vfs_write operations to prevent the VFS nfsd_write procedure from being fully executed, which in effect emulates an infinitely fast file and disk I/O subsystem on the server. The second patch removes the RDMA transport from NFS's write-path to emulate an infinitely fast network. It short-circuits only the NFS3PROC_WRITE_RPC operation, and leaves the rest of the NFS, RPC, and XDR procedures to function as usual. Both short-circuits can be toggled on a running system through a switch implemented in /proc.

Additionally, we applied a kernel patch by James Schutt to test the effect of NFS RPC transfer-size (rsize/wsize) on performance. By default, RDMA transport, iWARP as well as IB, uses a 32 KB RPC payload size. The only way to implement larger RPC requests in the 2.6.25 and 2.6.26 kernel is by increasing the value of RPCRDMA_MAX_DATA_SEGS and then recompile. Note that a bug in the RDMA transport code under the 2.6.25 and 2.6.26 kernel mandates the patch in http://marc.info/?l=linux-nfs&m=121936891515202&w=2 to be applied prior to this patch in order to make this work correctly. Also because many stack variables in the kernel are dimensioned by RPCRDMA_MAX_DATA_SEGS, using larger than 64 KB RPC's would require significant work. We only developed a simplified patch to provide proof of concept.

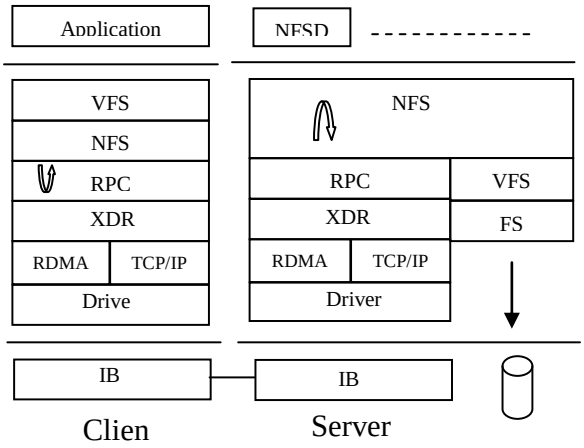


Figure 4, NFS Write Path Traversing the Protocol Stack from Client to Server

All patches developed in this study can be found in [12].

2.3.2 Server–side Short-circuit

Table 1 summarizes our benchmark results with the server-side short circuit applied, and with varying record as well as RPC transfer sizes. As shown, both the 512KB and the 1MB record-size had a negative impact on throughput, presumably due to the overhead of segmentation/reassembly to match the RPC Transfer size on the client and server. Increasing the RPC [13] transfer size from 32 KB to 512 KB, however, improved the write performance, but throughput still maxed at 425MB/s, ~20% of the available IB bandwidth.

Table 1 Streaming Write Throughput with Server Side Short Circuit and Varying RPC Transport Sizes

RPC Payload (Bytes)	Throughput (MB/s)		
	32KB Record	512KB Record	1MB Record
32768	245.60	283.40	281.60
65536	377.00	350.50	293.00
131072	387.50	363.50	306.00
262144	401.40	335.80	305.00
524288	425.00	376.50	312.50

2.3.3. Kernel Profiling

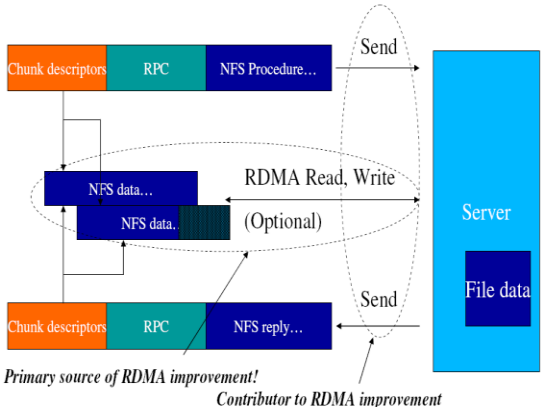
We enabled Oprofile [14] on the 2.6.26 kernel (including VMM, nfs and ib drivers) to search for bottlenecks that would explain why, with the short of NFS write on the server in place, we still only achieved 425MB/s write rates when the available bandwidth is 2 GB/second on the IB network. It is well to remember that Oprofile is primarily measuring CPU usage in a dual core system and that in x86 derived systems the CPUs may be taken on and off line by other hardware on the motherboard or the kernel itself. Thus, accounting for all CPU time is not equivalent to accounting for all wall clock time. Our results do not point a finger at any particular function as a bottleneck in the kernel. In several areas of the kernel Oprofile collects only incomplete data; in these areas samples are dropped reliably, independent of the buffer sizes or sampling rates used to collect profile data. The call graph visualization of the profiling data during a write experiment illustrates the deeply nested and massively tangled nature of the code paths that attempt to balance file I/O and other processes on the commodity multicore Linux machine. The Oprofile Summary Report, however, did report over 20% of total elapsed time spent copying data from user to kernel space.

With the server-side short circuit applied, we ran iozone using an 8 MB record and toggled between bypassing (with o_direct) and not bypassing (without o_direct) data-copying.. Without "o_direct" to bypass data-copying, Oprofile reported 22% of elapsed time spent copying data, versus zero time

with "o_direct", with a throughput improvement from 425 to 539 MB/s, a 28% gain.

2.3.4. RPC RDMA Transport

Figure 5 depicts the RDMA mechanism employed by the NFS RPC transport. As shown, large data segments are chunked into Linux page size (4 KB) first. A descriptor consisting of pointers to these pages is then generated and pre-pended to the RPC write request before its transmission. Upon receipt of this request, the server's RPC layer passes the client-side chunk descriptor to its IB hardware for RDMA-reads of page-size data chunks from client to its buffer cache; RDMA operations on page-size chunks is not at all optimized considering the RDMA overhead. In this case, the 32KB RPC data payload required 8 individual RMDA transfers instead of 1. Because today's IB hardware doesn't support scatter-and-gather to handle discontinuous physical memory, RPC is limited to transferring content of page-size memory locations per RDMA operation. A patch [15] is available through the NFS kernel community that implements a software solution before the next generation IB hardware becomes available. We didn't pursue the software solution in our study because our preliminary result demonstrated very little performance gain; we believe the RPC RDMA mechanism, though not efficient, is not the performance bottleneck in the NFS write-path at the moment.



.

Figure 5, NFS RPC RDMA Mechanism

2.3.5. Client-side Short-circuit

To further isolate NFS's performance bottlenecks, we applied a client-side patch that stubs the RPC write and removes the RPC RDMA transport from the write-path. With this patch, we hope to understand the impact of interactions between the Linux Virtual File System, Network File System, and Virtual Memory Management on NFS's streaming-write performance. Figure 5 plots the results of an iozone test suite in 3-D, with the x-axis reflecting the varying record size in KB, y-axis the iozone throughput in KB/s, and z-axis the file-size in KB. This experiment evaluated the effects on throughput of record-sizes ranging from 4K to 16M Bytes, and file-size ranging from 64K to 2G Bytes. As predicted, we found 32KB the optimal record-size because it matches RPC's default payload size. Additionally, we found that file sizes less than 256 Mbytes can yield up to 1.25GB/s of throughput; however, performance dropped drastically beyond this point to 700MB/s, and gradually to 500MB/s as the file-size reaching 2 GB, reflecting the heavy impact on streaming-write due to the interactions among Linux's VFS, NFS, and VMM. These complex interactions apparently impose a limit on further improvement, which is substantially less that the theoretical streaming bandwidth of the fast interconnects.

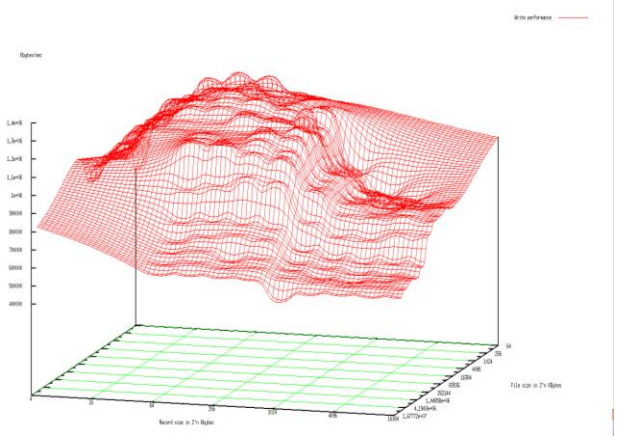


Figure 5, Results of IOZONE Test Suite in 3-D

3. Summary Results

The section summarizes the improvements we achieved on NFS's streaming-write using optimization techniques developed throughout this work. Additionally, we repeated our benchmark without the server- and client-side short-circuits in order to verify that these techniques also improve NFS's end-to-end performance, from user application on the client to file and disk I/O on the server. A summary of our improvement is presented in Figure 6. As shown, we obtained 130 MB/s of end-to-end throughput from a previous study with the NFS over RDMA distributed with the 2.6.16 kernel. The kernel community subsequently improved the performance to 212 MB/s through bug fixes and advanced features in the 2.6.26 kernel. We increased the RPC RDMA payload from 32KB to 512KB for better network efficiency, which further improved the write rate to 250MB/s. Because Oprofile revealed heavy penalties copying user data to kernel buffer, we experimented

with o_direct during streaming-write, and were able to increase the throughput to 315 MB/s.

Figure 6 includes the short-circuit results which strongly suggest that the bottleneck is not on the server or the RDMA data transfers. We applied the server short-circuit to emulate infinitely fast server file and disk I/O, and achieved 425MB/s with the default 32KB RPC payload and without o_direct; with o_direct, on the other hand, our throughput reached 512 MB/s. With the server patch applied, we then increased the RPC payload to 512KB, and were able to achieve 469MB/s without o_direct, and 528MB/s with o_direct. Because 528MB/s is still only ¼ of the network's available bandwidth, we removed the RDMA transport from NFS's write-path to investigate whether RPC RDMA is the performance bottleneck. With the client side short circuit, we repeated the benchmark without o_direct and achieved only ~500 MB/s, even though the memory bandwidth is 2.7 GB/s, reflecting a performance bottleneck in the interactions on the NFS client among the Linux VFS, NFS, and VMM when user data begins to full the VFS buffer cache.

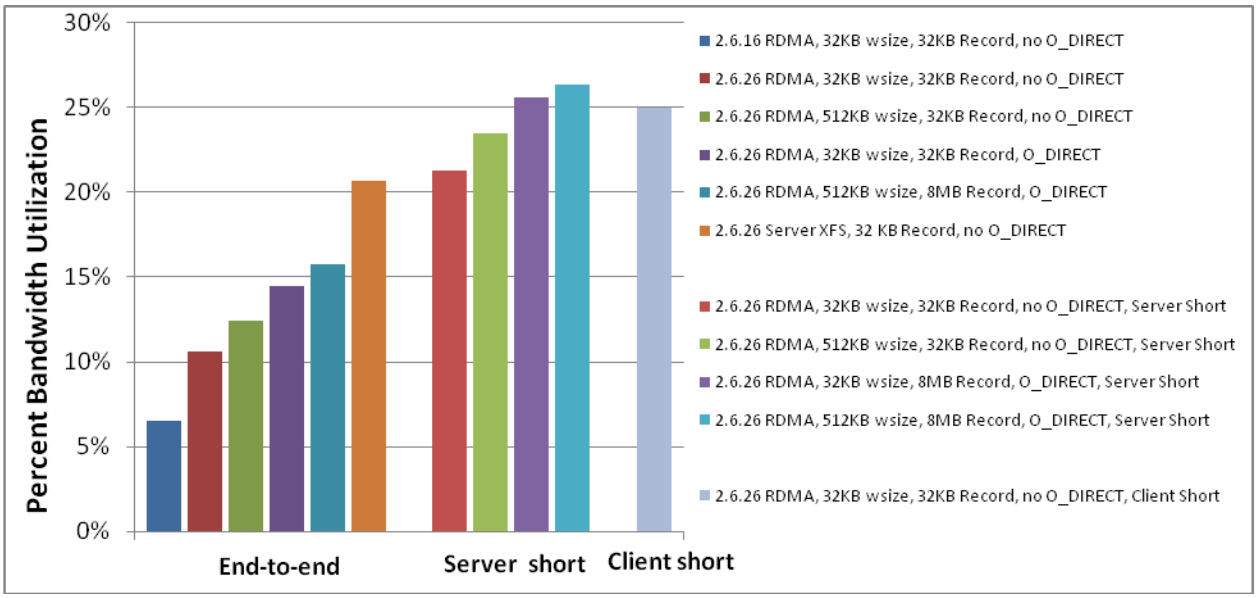


Figure 6, Summary of bandwidth measurements

4. Conclusion and Future Work

The goal of this project was to increase the efficiency of a high-speed, low-latency network with offload data transfer technologies such as NFS/RDMA and pNFS. In this we were successful to a large degree; the accumulated changes improved increasing efficiency in excess of 30%. The experiments that "short-circuited" the actual wire transfers showed that the bandwidth of the transport software stack on the client-side, alone, is limited to something on the order of 500 MB/s and, so, we have achieved a substantial improvement. However, even with this significant improvement, it is still only a meager fraction of the fabric capability.

The lack of a real improvement when employing RDMA seems reasonably attributed to the IB kernel stack. Future work could focus on that stack. In user-space this stack has proven able to garner nearly all of the wire bandwidth. Why, in kernel space, this constriction appears should be explained and remedied. However, it should be noted that our experiments in this direction do not entirely exclude the client stack. The fast-path hook for RDMA happens at a relatively deep point in the stack. Perhaps pNFS could better leverage the RDMA capabilities of the hardware as its fast-path hook is far higher than the RPC transport.

5. Acknowledgement

We'd like to express our appreciation to Tom Tucker from Open Grid Computing, Tom Talpey from Network Appliance, and James Schutt from Sandia National Laboratories for sharing their Kernel expertise.

6. References

[1] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, Third Edition: The Heart of Linux 2.6--From I/O Ports to Process Management" O'Reilly, November 22, 2005.

[2] William T. Futral, "InfiniBand Architecture: Development and Deployment – A Strategic Guide to Server I/O Solutions", Intel Press, August, 2001.

[3] R. Reici, P. Culley, D. Garcia, and J. Hilland, "An RDMA protocol Specification", Draft-ietf-rddp-rdmap-01, April, 2004.

[4] Dean Hildebrand, Lee Ward, and Peter Honeyman, "Large Files, Small Writes, and pNFS", Proceedings of the 20th ACM International Conference of Supercomputing, June 2006.

[5] B. Halevy and B. Welch, "Object-based pNFS Operations", Draft-ietf-nfsv4-pnfs-obj-12, December, 2008

[6] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, Third Edition: The Heart of Linux 2.6--From I/O Ports to Process Management", O'Reilly, November 22, 2005, Pp. 328-371.

[7] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, Third Edition: The Heart of Linux 2.6--From I/O Ports to Process Management", O'Reilly, November 22, 2005, Pp. 158-193.

[8] Helen Y. Chen, et al, "NFS over RDMA – IB and iWARP", OpenFabrics Sonoma Workshop at, May 1, 2007

[9] http://collectl.sourceforge.net/

[10]Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, Third Edition: The Heart of Linux 2.6--From I/O Ports to Process Management", O'Reilly, November 22, 2005, P. 330.

[11] http://www.iozone.org/

[12] http://sourceforge.net/projects/hpcfs/

[13] R. Srinivasan, RPC: Remote Procedure Call Protocol Specification Version 2, IETF RFC 1832, August 1995.

[14] http://sourceforge.net/projects/oprofile/

[15] git://git.linux-nfs.org/projects/tomtucker/xprt-switch-2.6.git.