Adaptive Memory System over Ethernet

Jun Suzuki, Teruyuki Baba, Yoichi Hidaka[†], Junichi Higuchi, Nobuharu Kami,
Satoshi Uchida[††], Masahiko Takahashi, Tomoyoshi Sugawara, and Takashi Yoshikawa

System Platforms Research Laboratories, NEC Corporation *[†]IP Network Division, NEC Corporation*
[††]2nd Computers Software Division, NEC Corporation
{j-suzuki@ax, t-baba@ax, y-hidaka@bq, j-higuchi@ax, n-kami@ak, s-uchida@ap, m-takahashi@ex, tom-sugawara@ap, yoshikawa@cd}.jp.nec.com

Abstract

For cloud computing, computer infrastructures need to be scaled up adaptively. However, their local memories cannot be expanded beyond the amount loaded to each computer. We present a method for scaling up of memory system beyond its local memory's capacity by high-speed page swapping using an adaptively attached solid-state disk (SSD) to a computer. Our PCI Express (PCIe) technology, "ExpEther" (Express Ether), interconnects a computer and a PCIe-based SSD via a standard Ethernet. The data transfer between the local memory of the computer and the SSD is performed without slow TCP/IP but with PCIe-standard direct memory access (DMA). It achieves IOPS of 33-K read and 36-K write for an access of 4-KB page size, which is twice as good as that for iSCSI with TCP-offloading. With the proposed method, a computer which only has 2-GB local physical memory can sustain its performance even when a 10-GB in-memory database is loaded.

1. Introduction

For cloud computing, computer infrastructures need to provide computing resources adaptively, in accord with resource utilization. However, one of the most important resources, local memory, cannot be shared among individual computers. This leads to a situation in which allocations to individual computers will be made on the basis of their respective needs at peak utilization times. This results in over-provisioning for most of their operational times.

This problem can be overcome by adaptively attaching external memory resources to a computer and expand its total memories. Some studies have reported sharing the local memory resources of individual computers using a technique of remote paging [1, 2, 3] or global management [4, 5]. With these methods, a bottleneck of the performance exists in memory management performed in software, and also in communication performed in such slow protocols as TCP/IP. Other studies have reported assigning external memory resources to a computer from special memory modules, for use as a second memory [6, 7, 8]. These methods have a potential to realize adaptive memory attachment in a relatively small performance-overhead. However, it needs to overcome implementation hardship in software and hardware. For hardware, a memory module is hard to place in a network. And also for software, special treatment is necessary for such memories to hide the overhead of access latency over the network.

Today, flash memories appear to hold great promise for complementing local memories from the access speed point of view [9, 10], and it is a good candidate for use in memory expansion. However, it usually provides only block access as a solid-state disk (SSD) with standardized local bus interfaces such as PCI Express (PCIe) and serial ATA, which are for use only inside a computer.

In this paper, we follow the external memory modules approach, but realize high-performance adaptive memory expansion with less implementation hardship and evaluate its performance. In our proposed method, we adaptively attach a PCIe-based SSD on Ethernet to a computer by page swapping. The performance test reveals it can sustain 10GB-in-memory database for a computer equipped with only 2-GB local memory.

The remainder of this paper is organized as follows. In Section 2, we present our method for adaptively attaching an SSD to a computer for memory expansion. In Section 3, we discuss our experimental results. We discuss future works in Section 4 and summarize our study in Section 5.

2. Attaching SSD using Ethernet

We propose an adaptive memory system where we attach a PCIe-based SSD to computers and use it for page swapping. This allows us to dynamically scale up computer systems beyond their local memories. We adopt existing PCIe over Ethernet technology, "ExpEther" [11] which interconnects a computer and a PCIe-based SSD via an Ethernet. It provides two important features for the memory system:

- **DMA over Ethernet:** We perform the data transfer between the local memory of a computer and an SSD without TCP/IP but with PCIe-based direct memory access (DMA) over an Ethernet. It enables us to perform high-speed IOPS in a small

access size sutable for use in page swapping which is performed in 4 KB in many platforms.

- **Hot-Plug and Remove over Ethernet:** By emulating PCIe-compliant hot-plug and remove functions over an Ethernet, we are able to adaptively attach SSD resources without stopping computer operations.

These technologies make it possible to perform high-speed page swapping and adaptive attachment of an SSD using commercially available Ethernet switches and PCIe-based SSDs. Although other interconnection technologies such as InfiniBand can perform the similar I/O attachment function [12], the special implementation to a host bus adaptor and an SSD is needed to adapt them to their high-speed interconnection specifications. In the following subsections, we discuss our over-network DMA and hot-plug in detail.

2.1. DMA over Ethernet

Our method performs PCIe-based DMA between the local memory of a computer and an SSD over an Ethernet. Because it skips intermediate protocol processing and memory copying of the transferred data, it provides higher-speed data transfer over network than those of conventional protocols.

To perform PCIe-based DMA over an Ethernet, ExpEther provides two functions shown in Figure 1: (1) it extends the PCIe tree of each computer to an Ethernet, functioning as a PCIe switch; (2) it performs lossless, low-latency transmission between ExpEther bridges located in a computer-side and SSD-side [13].

Extending PCIe Tree to Ethernet: A PCIe switch is a device which splits a PCIe bus in order to connect a computer to multiple I/O devices. A pair of ExpEther bridges and Ethernet performs a comparable function of a single PCIe switch between the computer and I/O devices, by the emulation of response of a PCIe switch in ExpEther bridges. Since Ethernet transport part is transparent to OS, ExpEther is able to extend the PCIe tree of the computer to an Ethernet-attached I/O device, which here is an SSD, without the need for any additional device driver or OS modification.

Lossless Low-Latency PCIe Packet Transmission: PCIe system assumes packet loss during transmission not to occur, and its performance depends on the latency of PCIe paths. For these reasons, we make ExpEther bridges perform lossless transmission and congestion control of encapsulated PCIe packets in an end-to-end manner. The lossless transmission function resends frames lost during transmission, guaranteeing the transmission of frames encapsulating PCIe packets. The congestion control function minimizes queuing delay at Ethernet switches. It employs delay-based sending-rate control. It monitors round-trip times (RTTs) between ExpEther bridges and adjusts sending rates to minimize RTTs. The result is suppression of queuing at Ethernet

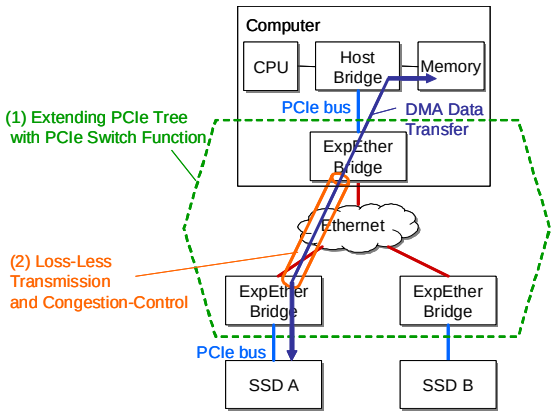


Figure 1. DMA data transfer using ExpEther.

switches, and minimizing transmission delay between the ExpEther bridges. The simulation result we reported in [13] showed the transmission latency of our mechanism was less than 8.5% of that of TCP which adopts an algorithm based on packet loss.

2.2. Hot-Plug and Remove

Since the standard PCIe protocol mainly focuses on I/O buses inside a computer and it is incapable of cluster-wide I/O resource management where several computers are included in a cluster, with ExpEther, we perform the separate management for the allocation of I/O resources and their attachment to computers. The individual computer to which a given endpoint is to be connected is determined by the VLAN. As may be seen in Figure 2, ExpEther bridges for endpoints connecting to the same computer are assigned a VLAN number corresponding to that computer. To change a connection for a given endpoint, a system manager alters the VLAN number assignment.

For triggering the attachment and detachment of I/O resources, ExpEther bridges periodically broadcast keep-alive frames within individual VLAN groupings. When a frame broadcast by the ExpEther bridge of an endpoint not currently attached to a computer is received, the receiving computer-side bridge interrupts the computer and begins a PCIe-compliant hot-plug process. By way of contrast, when no information frame has been received for a certain period of time, the computer-side ExpEther bridge interrupts the computer for a hot-remove process. When an interconnected SSD is used for a swap device, the hot-remove of an SSD should be performed when data is not swapped to the device. In an uncoordinated event, a process whose data is placed on the removed SSD needs to be killed and restarted afterward.

3. Experimental Results and Discussion

We performed two evaluation experiments. In the first experiment, we measured block I/O performance. In the

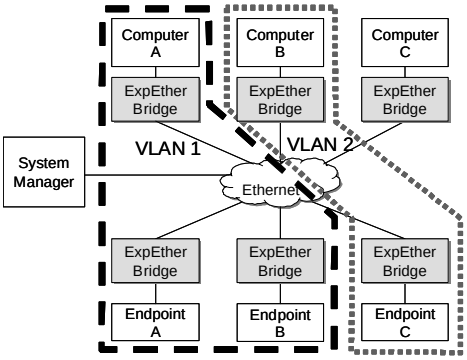


Figure 2. Grouping endpoints among VLANs.

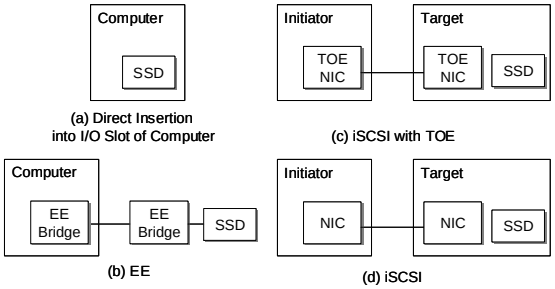
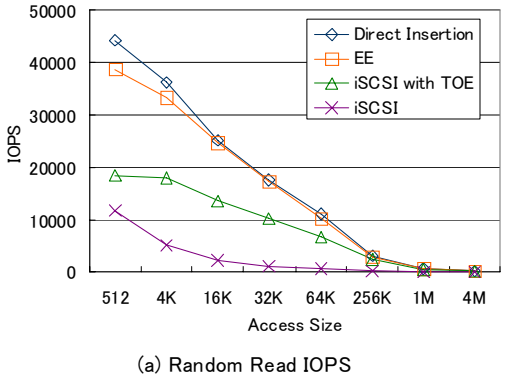
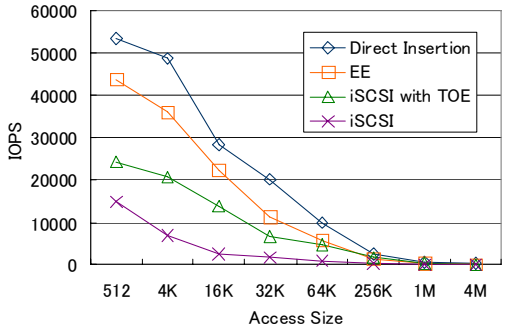


Figure 3. Experimental setups. EE denotes ExpEther.



(a) Random Read IOPS



(b) Random Write IOPS

Figure 4. Block I/O performance. (a) Random read I/O per second (IOPS). (b) Random write IOPS.

Table 1. Experimental environments.

Experiment	Block I/O	Database
CPU	Intel Core 2 2.66GHz	Intel Core 2 Quad 2.83GHz
OS	Cent OS 5.3 kernel-2.6.18	
Ethernet	10GbE	
SSD	16-GB partition of 160-GB Fusion IO SSD	

Table 2. IOPS for 4 KB access. Performance values are normalized to Direct Insertion.

	(b) EE	(c) iSCSI w/ TOE	(d) iSCSI
Ran. Read	92	50	14
Ran. Write	74	42	14
Ran. Read w/ Switch	91	46	14
Ran. Write w/ Switch	68	39	14

second experiment, we were able to show that a computer maintains its application performance even when it is consuming more memory than that available at its local memory slots.

The experiments were performed for the four setups shown in Figure 3: (a) an PCIe SSD used as a swap device is directly inserted to the I/O slot of a computer; (b) an SSD is connected to a computer by ExpEther (EE) bridge through DMA over Ethernet; (c) iSCSI is used as the interconnection protocol between a computer and an SSD (at the target, the SSD is inserted into an I/O slot, and the initiator and the target use a network interface card (NIC) with a TCP-offload engine (TOE)); and (d) iSCSI is used with an initiator and a target, employing an ordinary NIC.

Experimental setup (a) was used to determine a baseline for DMA data transfer to and from an I/O inserted into the I/O slot of the computer. Experimental setups (c) and (d) were measured as a reference of the performance of conventional protocols. Table 1 summarizes the specifications of the computers used in the experiments. The sum of the latency in the packet forwarding of a computer-side and an SSD-side ExpEther bridge was 1.45µs, while the latency in the inserted 10GbE switch was 0.92µs. Our prototype ExpEther bridge was implemented in a Field Programmable Gate Array (FPGA).

3.1. Block I/O Performance

We measured the block I/O performance of the SSDs using Iometer [14]. Figure 4 shows the IOPS performance for the four setups shown in Figure 3. Table 2 summarizes IOPS performance for the 4-KB access used for data transfer in swap operations in the proposed method. It also shows the results when a standard Ethernet switch was inserted between a computer and an SSD. The values are normalized to that of the direct insertion. Read performance with our method is nearly

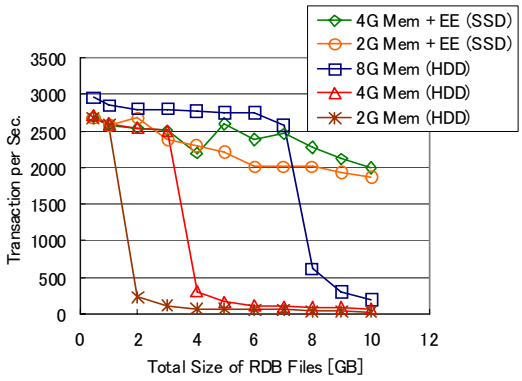


Figure 5. pgbench performance.

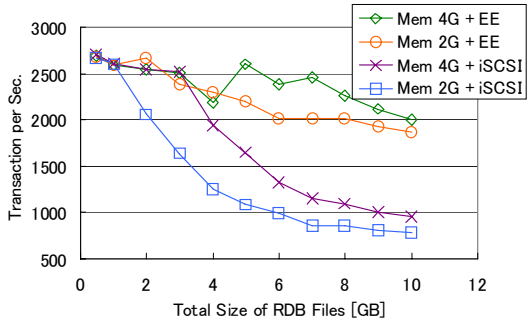


Figure 6. pgbench performance for EE and iSCSI.

the same as that with direct insertion (less than a 10% difference) and almost twice as good as that for iSCSI with TOE. While write performance was roughly 30% worse than that with direct insertion, it was still roughly twice as good as that for iSCSI with TOE. The performance was almost the same when an Ethernet switch was inserted.

The experimental results for block I/O performance show that our method successfully provides the benefits of high-speed SSD access by employing DMA data transfer. It also provides better performance than does a conventional protocol for sharing storage resources in a network. This is because our proposed method can skip protocol processing and memory copying which are performed by conventional protocols.

We analyzed 30% degradation in write performance by monitoring the PCIe traffic between the computer and the SSD. We found that, in the write access, the DMA controller in an SSD sends up to four memory read requests and waits to send the next one until it receives the completion. With the increase of the number of requests sent by the DMA controller at one time, we would be able to decrease the interval and fill the link bandwidth with the transferred data.

3.2. Application Performance

In this subsection, we show how a computer is able to maintain its application performance even when it is consuming more memory than that available at its local memory slots. As a benchmark application, we use Relational Database (RDB), with the RDB files stored in a ramdisk which resides in the local memory of a computer to demonstrate large memory consumption.

We used postgresql 8.1 for an RDB platform and a TPC-B-like pgbench [15] for a benchmark tool. We gradually increased the total size of RDB files which were placed in a ramdisk. When the total size of database files exceeded the size of the local memory, portions of the files began to be swapped out to a swap device. We performed the experiment using the SSD as a swap device in the previously mentioned 4 setups. With the setup (c) (iSCSI with TOE), unfortunately, we were unable to measure performance due to TOE software failure.

We first measured how much our method is capable of expanding memory. Figure 5 shows the performance of our method (setup (b) in Figure 3) and the HDD base lines for varying size of RDB files. We tested our method for local memory sizes of 4 and 2 GB, and the Ethernet-attached SSD was used as a swap device. For the baselines, the memory sizes were 8, 4, and 2 GB, and a local HDD was used as a swap device. With respect to the baselines, when RDB files started to be swapped out to the local HDD, performance decreased steeply because of the low access speed of the HDD. With our method, there was less performance degradation, for both 2 and 4-GB local memories, when RDB files started to be swapped out to the SSD. The average swap-in and swap-out for the 10-GB file size were 100 MB/s and 42 MB/s for the 2-GB-memory case, and 78 MB/s and 67 MB/s for the 4-GB-memory case. The performance when the SSD was directly inserted to the I/O slot of the computer (setup (a)) was same as our method. These results show application performance is sustained with our method even when the computer is consuming more memory than that which is available at its local memory slots. The results in Figure 5 also indicate the effectiveness of local memory reduction. For 6-GB RDB files, performance results achieved with our method in the 4-GB local memory case (33% memory reduction) were only 13% worse than those for the 8-GB baseline, and our results for the 2-GB case (66% reduction) were only 27% worse.

We next measured how much our method outperformed a conventional protocol in the sharing of storage resources in a network. Figure 6 shows the performance of our method and that of iSCSI (setup(d)). In the best cases, our method outperformed iSCSI by 113% with a 4-GB local memory and by 139% with a 2-GB local memory. These differences result from differences in data transfer methods. Our method uses DMA data

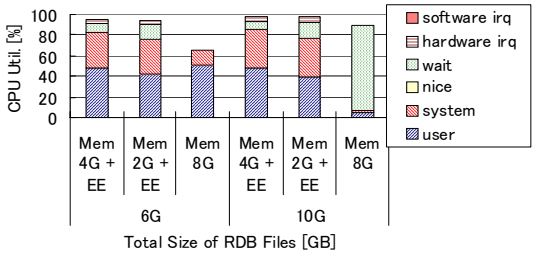


Figure 7. CPU utilization with pgbench.

transfer over an Ethernet, while iSCSI requires protocol processing and memory copying in order to transfer data. Although, as has previously been noted, we were unable to obtain performance results for iSCSI with TOE, the calculated results showed the performances for the case of iSCSI with TOE for the 10-GB file size were 1711 transactions per sec. for 4-GB local memory and 1541 transactions per sec. for 2-GB memory. In the calculation, we supposed that the expected number of accesses to the SSD per transaction was same among different setups and used the relative block I/O performances illustrated in Figure 4 to estimate the pgbench performance of iSCSI with TOE by the results of ExpEther and iSCSI.

The merits of memory expansion with the proposed method are also apparent from the standpoint of CPU utilization. Figure 7 shows CPU utilization levels with respect to cases when the local memory was set to 4 and 2 GB, and the Ethernet-attached SSD was used as a swap device. It also shows the case of the baseline with the memory size of 8 GB. The results shown here are for 10-GB RDB files and indicate that, with the proposed method, the CPU time was spent for application processing. On the other hand, in the 8-GB-memory baseline case, most of the CPU time is spent waiting for the completion of I/O requests for swap operations.

4. Future Works

In our work, we attach an SSD to a single computer and it is exclusively used by its computer. Ethernet-connected SSD resources should be shared among multiple computers for efficient hardware utilization. This would be enabled by using recently standardized PCIe I/O virtualization where an I/O device can process multiple contexts [16]. Also, we use one SSD for a swap device. Multiple devices could be used to increase the capacity and reliability. The multiple devices can be coordinated at device driver level using a RAID method or at OS level to coordinate data placement and I/O device assignment.

5. Conclusion

In this paper, we have presented adaptive memory system. It attaches a PCIe-based SSD to a computer over Ethernet by page swapping to scale it up beyond its local memories. The proposed system with our ExpEther technology enables high-speed page swapping and adaptive attachment of an SSD using commercially available Ethernet switches and PCIe-based SSDs. The block I/O performance is twice as good as that for iSCSI with TOE, which indicates higher speed swapping than conventional protocols. Further, application benchmark of a database shows we can sustain the performance when 10-GB in-memory database is loaded to a computer equipped with only 2-GB memory. These evaluation results show we can realize both high-performance adaptive memory expansion and less implementation hardship with our proposed method.

Acknowledgements

We would like to thank Y. Hasebe, H. Shimamoto, and K. Egarashi for their helpful suggestions. We also thank K. M. Fujita for his cooperation to experiment. This work was partly supported by Ministry of Internal Affairs and Communications (MIC).

Reference

[1] L. Iftode *et al.*, "Memory Servers for Multicomputers", *Compcon Spring '93,* pp 538-547, 1993.

[2] E. P. Markatos *et al.*, "Implementation of a Reliable Remote Memory Pager", *USENIX ATC.1996.*

[3] T. Newhall *et al.*, "Nswap: A Network Swapping Module for Linux Clusters", *Euro-Par 2003 Parallel Processing,* vol. 2790, pp 1160-1169, 2004.

[4] M. D. Dahlin *et al.*, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance", *USENIX OSDI,* no. 19, 1994.

[5] H. A. Jamrozik *et al.*, "Reducing Network Latency Using Subpages in a Global Memory Environment", *ASPLOS-7,* pp 258-267, 1996.

[6] K. Lim *et al.*, "Disaggregated Memory for Expansion and Sharing in Blade Servers", *ISCA-36*, pp 267-278, 2009.

[7] K. Li *et al.*, "Evaluation of Memory System Extensions", *ISCA-18,* pp 84-93, 1991.

[8] M. Ekman *et al.*, "A Cost-Effective Main Memory Organization for Future Servers", *IEEE IPDPS'05.*

[9] T. Kgil *et al.*, "FlashCache: A NAND Flash Memory File Cache for Low Power Web Servers", *CASES 2006,* pp 103-112, 2006.

[10] Intel Turbo Memory with User Pinning, http://www.intel.com/design/flash/nand/turbomemory/

[11] J. Suzuki *et al.*, "ExpressEther – Ethernet-Based Virtualization Technology for Reconfigurable Hardware", *IEEE HOTI'06* , pp 45-51, 2006.

[12] InfiniBand Architecture Specification, Release 1.1.

[13] H. Shimonishi *et al.*, "A Congestion Control Algorithm for Data Center Area Communications", *2008 Int. CQR Workshop*.

[14] Iometer project: http://www.iometer.org/

[15] pgbench, http://www.postgresql.org/docs/current/interactive/pgbench.html

[16] "Single Root I/O Virtualization and Sharing Specification Revision 1.1", PCI-SIG, 2010.