Cost Effective Data Center Servers

Rui Hou[1], Tao Jiang[1], Liuhang Zhang[1], Pengfei Qi[1], Jianbo Dong[1],
Haibin Wang[2], Xiongli Gu[2], Shujie Zhang[2]
[1]Institute of Computing Technology, Chinese Academy of Sciences
Email:{hourui, jiangtao, zhangliuhang, qipengfei, dongjianbo}@ict.ac.cn
[2]Shannon Laboratory, Huawei Technologies Co., Ltd
Email:{benjamin.wanghaibin, guxiongli, shujie.zhang}@huawei.com

Abstract

The exploding growth of digitalized information has led to the rapid growth of data centers, both in numbers and in size. Cluster has been the dominating system architecture used in most data centers. However, the increasingly diversified data center applications have requirements beyond what the cluster architecture can deliver. For instance, clouding computing requires flexible sharing of all data center resources. Big data applications often need large memory capacity. A few applications can use GPGPU effectively. Existing system might be extended to a certain degree to meet those needs. Those extensions however would often be prohibitively expensive. The paper presents our attempt to design a system using commodity products that can meet the varying needs of many emerging data center applications **in a cost-effective way**.

Our attempt is to create a system by connecting multiple nodes through a PCIe switch and then extend the software stack to support resource sharing among these nodes. In particular, a node can directly use the memory, NIC, and GPGPU of other nodes through the PCIe switch with no or little involvement from other nodes. We build a prototype as our evaluation platform. Our evaluation results indicate that those resources can be shared effectively in many cases. For using remote memory as block device, our prototype system has 5 times bandwidth, 11 times IOPS and 1/12 latency compared with the system connected by 10GigE in average for Orion benchmark; Using remote GPGPU via PCIe switch achieves average 60 times speedup than the case without GPGPU, and the performance loss is also acceptable (its average execution time is 1/3 of local GPGPU) for micro-benchmarks from GPU computing SDK; And using remote NIC via PCIe switch achieves average 95% bandwidth and 1.4 times latency of local NIC in httperf testing. While our prototype system offers multiple benefits, it is not perfect and has a lot room for further optimization and extension. We hope the outcome presented in this paper will encourage more researchers to join us in designing highly efficient and cost-effective servers.

1. Introduction

More and more organizations have built or are building data centers to provide services to the mass. Due to fierce competition, data center owners are particularly sensitive to the cost of data center servers. Traditionally, most data center servers use the so-called cluster architecture that connects low end servers through Ethernet or Infiniband (IB). Many traditional applications have been tuned to work with the cluster architecture. However, as applications become more diversified, the cluster architecture becomes inadequate in handling an increasing number of applications. To make matters worse, service providers often push out new applications before they can be optimized due to time-to-market pressure.

One of the key shortcomings of the cluster architecture is the difficulty to share resource among different nodes. Within a computer cluster, each node has its own set of resources, such as memory, GPGPU, and interconnect links. There is no effective way for one node to access the resources of other nodes. A traditional way of sharing resources of different computing nodes has been using the high-end SSI (single system image) system, such as those high-end cc-NUMA HPC (high performance computing) systems. However, these systems, especially those large scale ones, command heavy price. As a result, they are rarely used by cost-conscious data center operators, who continue to favor clusters. To better meet the varying needs of their existing and future applications, data center operators simply provision some extra resources to provide some wiggle room for potential expansion. This approach has led some heavily over-provisioned servers in the data centers. It is thus highly valuable to find a cost-effective design that allows efficient, flexible resource sharing. Some companies, like Google and Facebook, have been building servers customized for their data center workloads. On the other hand, most companies do not have the resources to do what those selected few are doing. They have to purchase what is available on the market.

The purpose of our work is to seek for a solution that is low in the cost but high in the flexibility and efficiency of resource sharing. Our goal is to allow a node to dynamically use the memory, storage, network, and GPGPU physically located within other nodes in the system. For example, when a big data application needs more memory than the nodes it runs on can provide, it can simply "borrow" some unused memory from other nodes that are running CPU intensive applications without large memory footprints.

As the first attempt to achieve our goal, we built a proto-

978-1-4673-5587-2/13/$31.00 ©2013 IEEE

type which consists of low end servers connected via a PCIe switch with the non-transparent bridge support. We investigate the feasibility of using remote memory (i.e., memory of other nodes) through either direct load/store instructions or the DMA engine. We also study the potential of using remote GPGPUs and NICs. Our results reveal that: (1) Memory can be shared in multiple different ways. Any sharing through DMA requests performs wonderfully, while sharing through direct load/store may incur high overhead caused by the modern processors that contain no special optimization for cache line loads/stores through the otherwise considered pure I/O interface. For using remote memory as block device, our prototype system has 5 times bandwidth, 11 times IOPS and 1/12 latency compared with the system connected by 10GigE in average for Orion benchmark; (2) Sharing computing capability offered by GPGPU has huge potentials. Using remote GPGPU via PCIe switch achieves average 60 times speedup than the case without GPGPU for micro-benchmarks from GPU computing SDK; (3) Using remote NIC via PCIe switch is practical, which achieves average 95% bandwidth and 1.4 times latency of local NIC in httperf testing. Also, for some cases which are inefficient for resource sharing, our work identify the potential bottlenecks and provide insights on it future optimizations.

In the rest of the paper, section 2 describes the PCIe Switch system we build. Section 3 presents the design of three case studies. Section 4 contains the evaluation results. Section 5 discusses the insights and issues from our experiments. Section 6 talks about related work. Section 7 concludes the paper.

2. Hardware prototype system

To keep the cost in check, we use exclusively commodity products for the prototype system. We select x86 processor chips from Intel for the computing nodes and PCIe switch chips from PLX for the interconnect. We have considered Intel QPI and AMD HT products but decided against using them because they are proprietary and do not have the widespread use as PCIe does. PCIe interface can be found in almost every major microprocessor, thus allows us to add almost any mainline microprocessor to the system.

Table 1: Interconnect comparisons.

	Peak bandwidth	Application Latency	Remote memory access	External Switch
PCIe switch PLX PEX 8648[1]	3005MB/s	∼1us	RDMA, direct access via load/store	Yes
10GigE	1175MB/s[2]	8.9us[3]	RDMA	Yes
Infiniband 40Gb/s Mellanox PCIe Gen2[3]	3400MB/s	∼1us	RDMA	Yes

[1] The numbers are from PLX PEX 8619 user manual[25].
[2] The numbers are from our test by using Netperf.
[3] The numbers are from Mellanox report[2].

The selection of the interconnect technology is the most critical decision for the hardware prototype, because the desired resource sharing will only happen through the selected interconnection links. The conventional choices are Ethernet or IB. To put the raw characteristics of PCIe in perspective, Table 1 compares the key features of PCIe Gen2, 10 Gigabit Ethernet (10GigE), and IB. PCIe and IB have similar peak bandwidths and the same latency, and 10GigE has the lowest peak bandwidth and a latency of an order of magnitude more than that of PCIe and IB. All three support high Remote DMA(RDMA) and need external switches in real deployments.

What makes PCIe unique are two features not seen with Ethernet and IB. First, PCIe allows direct access to the remote memory via normal load/store instructions. This feature allows a program to make use of remote memory without modification. It is supported by a PCIe switch with the Non-Transparent Bridge (NTB) functionality. NTB provides isolation among the hosts connected via a PCIe switch while still allowing communications among the hosts. Second, PCIe allows “shorter” communication channel. A typical communication channel with Ethernet and IB goes through the Ethernet and IB adapters connected to the north bridge via a PCIe interface. PCIe-based communication channel is shortened because the adapters and additional protocol conversion (from PCIe to IB or Ethernet) are cut off from the path.

Figure 1 shows the prototype system. The top half of the figure contains the block diagram of the system and the bottom half contains a picture of the backplane. The prototype system includes five computing nodes and a backplane. Each computing node is connected to the backplane via a PCIe adapter card. The backplane contains one transparent PCIe bridge chip and four NTB chips. The transparent bridge chip (PLX PEX 8648 [22]) supports one root node (CPU 0) and four leaf nodes (CPU 1-4), so the system has five nodes in total. It is in charge of forwarding transactions between root node and leaf nodes. The NTB chips (PLX PEX 8619 [21]) are used for address isolation and translation. The prototype supports PCIe Gen2 interfaces.

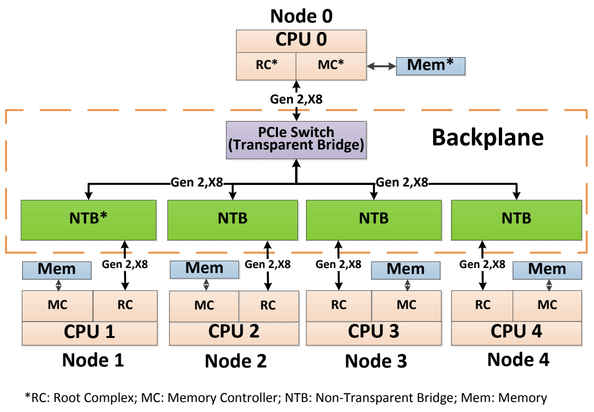
Table 2: Node machine configuration.

CPU	Intel i7 processor, 3.4GHz, 4 cores, 8 threads, 8MB LLC
Memory	4GB-8GB DDR3
Disk	SATA 7200RPM 1TB

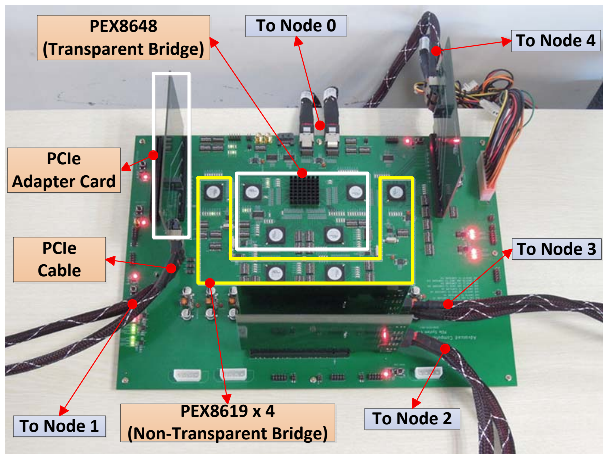
Any machine with a PCIe x8 slot can be connected to the backplane, our choice of the computing nodes is shown in Table 2. The root node and leaf nodes have the same configuration.

3. Resource sharing mechanisms

We design and implement three resource sharing mechanisms on the prototype system to allow all nodes to share all the memory, NICs, and GPGPUs installed in the system.



(a) Architecture



(b) Backplane

Figure 1: A PCIe switch based system.

3.1. Sharing memory

In the prototype system, the NTB chip supports either side directly access the memory of other side. It includes doorbell registers for sending interrupts from one side of the bridge to the other side, scratchpad registers accessible from both sides for inter-processor communications, and address translation support between the (different) memory address spaces of the two sides.

To set up memory sharing, the two sides first negotiate with each other to determine the memory regions that will be accessible by the other side. The side where a memory is located is called the host side. The other side is called the receiving side. Each side is responsible for setting up its mapping table in the address translation module in NTB. After the setup, one side can access the memory region of the other side (called *remote memory* hereafter) via either direct load/store instructions or the DMA engine located inside NTB.

3.1.1. Direct access At the direct access mode, applications do not have to be aware of the existence of the remote memory. They access the remote memory in the same way they access the local memory, even though the underlying hardware path is different. Taking a remote load that misses in all caches as an example, the load request first goes to the memory controller. The memory controller determines it is not a local address and passed it to the PCIe root complex (RC). The PCIe RC receives this request and converts it to a PCIe MemRd (Memory read) TLP (Transaction Layer Packet). This TLP with the given address is then sent to the PCIe switch chip. The switch chip routes this packet to the corresponding NTB chip. The NTB chip translates the given address into a remote physical address and sends the packet to the destination node. The "remote" physical address in the packet is in fact the local memory address of the destination node. After the destination node receives this address, it loads the most up-to-date data from the memory or one of its internal cache. It then constructs a completion packet with both data and acknowledge information. The completion packet is sent back to the requester node using the same path. A store operation has the same process, other than it does not need a completion packet.

We implement a driver hooked to the memory management module of Linux. The driver contains necessary support for managing and accessing the remote memory. It provides two function calls, remote_malloc and remote_free, for applications to allocate/release the remote memory. The driver is in charge of the initialization of the switch and NTB chips. During remote memory allocation, it chooses virtual address regions and creates corresponding page tables for the remote memory regions, and sets up the related MMIO registers of the switch and the NTB chip.

The mechanism essentially provides an illusion of a large pool of available memory. It has some similarity with cc-NUMA mechanism. One key difference is that a memory region in cc-NUMA can be accessed simultaneously by all nodes, while *a memory region in our design can be accessed by only one node at any given time*, even though it can be accessed by different nodes at different times. In addition, during our experiments, we found that the Intel processors (including Xeon and i7) do not allow lock prefix instructions to be used on the remote memory, which implies the remote memory space cannot be used for lock/barrier variables or any atomic data structures that require atomic operations.

3.1.2. DMA access The DMA access mode uses the DMA engine located inside NTB to perform bulk data transfers from/to the remote memory.

Both block DMA and burst DMA are supported by the NTB chip (PLX PEX 8619). In the block DMA mode, a single descriptor is programmed and sent to the DMA engine. A DMA descriptor contains the configuration of a DMA transfer, such as the source address, the destination address, transfer size, etc. An interrupt signal is raised after the specified DMA task completes. Only after processing the interrupt signal, the CPU will send the next DMA task. Then in block mode, the software essentially launchs DMA tasks serially. The burst DMA mode, on the other hand, can launch DMA tasks in batch. It allows multiple descriptors, one for each DMA task, to be programmed into the DMA engine simultaneously. Without surprise, the burst mode can achieve much higher sustained bandwidth than the block mode.

Figure 2 shows how these two modes perform when there are two pending DMA tasks, and each task consists of two TLPs. Even with a limited number of parallel DMA tasks, the burst mode is much better than the block mode. Although the TLPs which belong to the same DMA request can be pipelined, the block mode has to do interrupt handling each time a DMA transfer finishes. The burst mode requires only one interrupt handling at the end of last DMA request.

To simplify the use of the burst mode, we implement a virtual block driver (VBD) in Linux to emulate a block device backed by the remote memory, as shown in NodeA of Figure 3. The VBD transmits data from/to the remote memory using the burst DMA mode. It includes an active request queue and an waiting request queue. A newly arrived block request from the I/O scheduler is inserted in the tail of the wating queue. If the active queue is not empty, the DMA engine will launch a DMA transfer for all requests in the active queue. After the DMA transfer finishes, the active queue is cleared. When the active queue is cleared and the waiting queue is not empty, the role of these two queues will be exchanged.

To study the effectiveness of the emulated "block device", we use it as a disk or a swap partition: the receiving node uses the remote memory as its swap space.

3.2. Sharing GPGPU

Although GPGPU is a powerful accelerator for computing intensive workloads, only a limited set of applications can utilize it effectively. Our experience from building a large-scale GPGPU-based supercomputers tells us that having a few GPGPU nodes in system is a good thing because there are always some workloads that use some computing capability in data center. However, having many GPGPU nodes in a system is not near cost efficient because a vast majority of applications cannot use more than a few GPGPU's effectively.

Our goal here is to explore if installing only a few GPGPU card in a system and then allowing everyone in the system to use these GPGPU cards makes sense. To allow a GPGPU

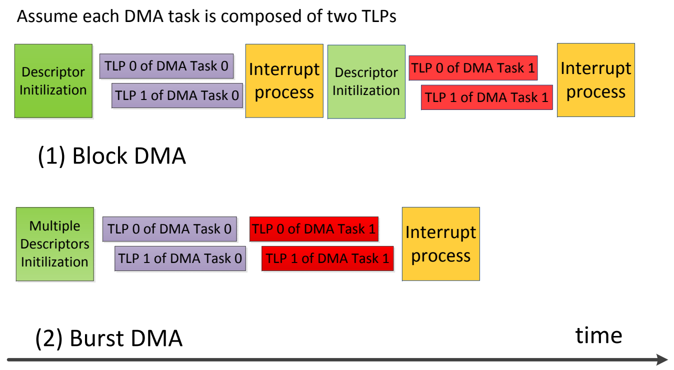


Figure 2: PCIe Switch based system

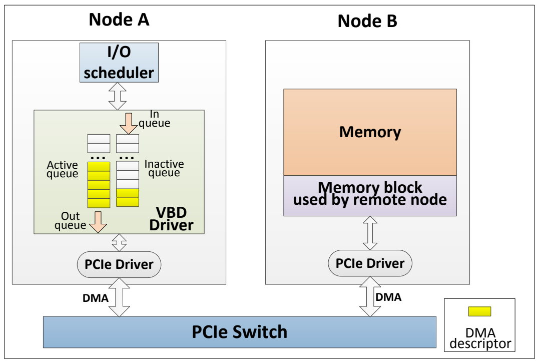


Figure 3: Virtual Block Device

connected to a host node through the PCIe interface, the host node must first expose a memory region to other nodes. This region contains five parts: (1) the binary buffer, used for storing executables to be run on the GPGPU, (2) an input data buffer, (3) an output buffer, (4) a task start flag, and (5) a completion flag.

When Node A wants to use the GPGPU located in Node B. Node A and Node B first negotiate, as discribed in Section 3.1, for Node A to get the ownership of the corresponding memory region of Node B. Node A then copies the GPGPU program to the binary buffer of the remote memory using DMA writes, and then notifies Node B to load this program into the GPGPU. The data to be processed is transferred into the remote input data buffer using the DMA engine. Then the remote task start flag is marked as valid after the input data is transferred. Node B copies data from the input data buffer to the corresponding GPGPU memory before starting execution.

After the execution, the output data will be moved into the memory region. and the completion flag will be set. Once Node A detects the completion by polling the completion flag, it will collect the output from the remote memory using DMA reads. Other completion detection mechanisms are possible, but are beyond the scope of this paper.

3.3. Sharing NIC

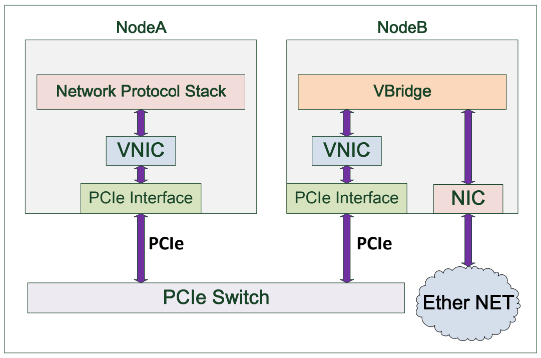


Figure 4: Using remote NIC

In order to allow existing applications to run on the prototype system transparently, we have implemented the IP layer over the PCIe protocol, which enables TCP/IP traffic to go through the PCIe links seamlessly. Specifically, we have developed a driver to emulate virtula NIC (VNIC). A VNIC has its own IP and MAC address and can co-exist with real NICs. The VNIC driver is invoked whenever there is TCP/IP traffic associated with the IP and MAC of a VNIC.

Each VNIC maintains a routing table that records mappings for any given MAC address to its associated PCIe link id. For a node pair that communicate via VNIC, a send queue and a receive queue are allocated in the memory of each side and the address translation module in the corresponding NTB chips is configured accordingly. Packets are copied from the send queue of one side to the receive queue of the other side using low-level PCIe switch functionality described in Section 3.1.

The baseline design uses the block DMA mode. Since, it is common for the send queue to have many pending requests, especially when network intensive applications are running. We add the burst DMA mode as an optimization. Their comparison is shown in Section 4.

With VNIC, each node in the prototype system is assigned with an internal IP and MAC address, and it can communicate with other nodes via TCP/IP protocol. An existing software net bridge (VBridge) in Linux is used to connect VNICs and real NICs, as shown in Figure 4. Node A first connects to Node B through VNIC backed up by the PCIe switch subsystem. The VNIC of Node B then connects to its NIC through the VBridge. With connections, Node A can use the NIC of Node B for its own purpose.

4. Performance evaluation and analysis

This section presents the performance results of our experiments done on the prototype system.

4.1. Remote memory

4.1.1. Basic evaluation Moving data from one node to another can be done either by loads initiated at the destination node or stores initiated at the source node. Consequently, there are different methods to move data across the nodes in the prototype system, including direct loads, direct stores, block DMA reads, block DMA writes, burst DMA reads and burst DMA writes. We use each method to transfer 1GB data between two nodes. The data is split into blocks that vary from 64 bytes to 1MB bytes. Figure 5 shows the measured bandwidth. We can draw four observations from Figure 5.

(1) Direct load has horrible performance and varying the bock size has no effect on its performance. As an attempt to understand this behavior, we used Tek PCIe Logic analyzer [6] to collect the TLP traces. In the 17us random sampled observation window shown in Figure 6, there are about 400 TLPs with DMA reads, but only 12 TLPs with direct load. With direct load, a TLP read request is issued only after its previous request is finished. The serial processing of the read requests severely impacts the performance. All other methods

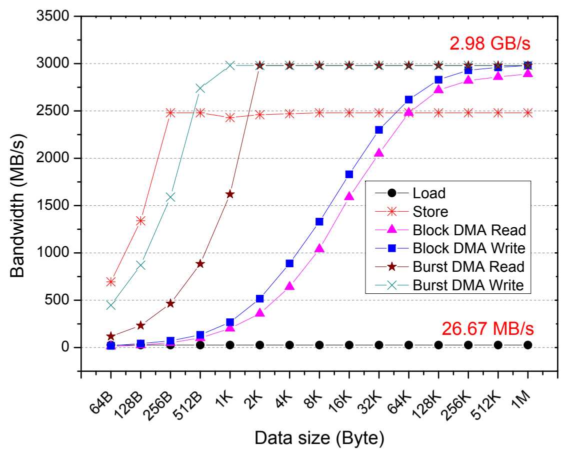


Figure 5: Achieved bandwidth

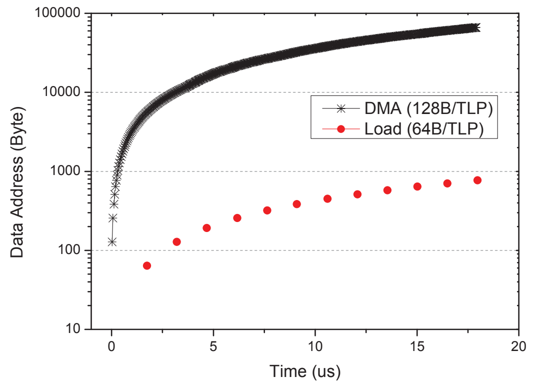


Figure 6: Observations by Tek PCIe logic analyzer

see pipelined parallel TLP requests and perform significantly better than the direct load method. We suspect the PCIe root complex (RC) inside the processor chip is the culprit that serializes load requests. Extending it to support parallel load requests is a potential optimization worth exploring.

(2) DMA transfers are in general more effcient than direct loads/stores. The theoretic node-to-node peak bandwidth of the prototype system is 3.05GB/s [21, 25]. All DMA transfers can achieve a near peak bandwidth with sufficiently large block sizes. One obvious reason is that DMA requests are sent in batches. In addition, a TLP packet can carry up to 128 bytes of data, so DMA mode issues exclusively 128-byte requests. While a refill request resulted from a cache miss of a direct load request asks for only a cache line worth of data – 64 bytes.

(3) Writes perform better than the equivalent reads. PCIe protocol uses non-posted transactions for reads and posted transactions for writes. In case of a non-posted transaction, the sender does not release the occupied resources until the acknowledge packet comes back. In case of a posted transaction, the sender releases the associated resources right after the

trasnaction is sent out. In addition, the processor is enabled to combine direct stores into large store transactions, making direct stores perform much better than direct loads.

(4) Without surprise, burst DMA performs better than block DMA, especially when the block size is less than 128K bytes, as it allows better pipelined less interrupted data movement.

4.1.2. Direct load/store mode Intel IA-32 architecture allows data loaded from the remote memory to be saved in the local caches [12]. However, we have found that writes to the remote memory must be carried out in the write-through mode in cacheable mode. Our tries of using the write-back mode on three different IA-32 processors all have failed. We are investigating this issue. At the write-through mode, a write always goes to the (remote) memory regardless of if it is a cache hit or a cache miss.

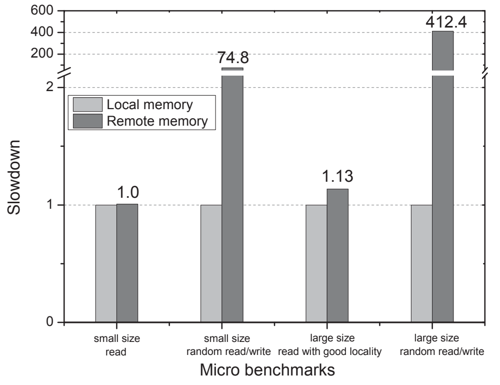


Figure 7: Performance evaluation for direct load/store access

To test effectiveness of caching remote data locally, we use four micro-benchmarks: (1) one with a read-only working set smaller than the LLC (last level cache); (2) one with a read/write working set smaller than the LLC; (3) one with a read-only working set larger than LLC but with good locality; (4) one with a random read/write working set larger than LLC without locality. The ensuing results are shown in Figure 7.

For case 1 and case 3, there is no performance difference between using the local memory and using the remote memory. This implies that storing load-rich data with good locality on the remote node will unlikely degrade the performance, even without improving the underlying PCIe Switch based system. For case 2 and case 4, using the remote memory is significantly worse than using the local memory. While the performance of case 4 is in line with our expectation, the performance of case 2 is a surprise.

Intel processor manual [12] states that a write at the write-through mode either refills (Option 0) or invalidates (Option 1) the matching cache lines. We had selected Option 0. To find out if our selection is in effect, we use a small test. In the test, host A reads a variable located in host B and then writes it with value 0, and then Host B write it with value 1. After that, A reads the variable again and gets value 1. If Option 0 were in effect, the variable would have been cached in A and A would have read value 0 because there is no hardware mechnism to maintain the coherence between A and B. Since A gets value 1 instead, we can conclude that Option 1 is being used. Please note the proposed way of using the remote memory does not have the coherency problem described here, because the host node is not allowed to access the part of memory it gives to the others. Studying the effect of Option 0 is ongoing work.

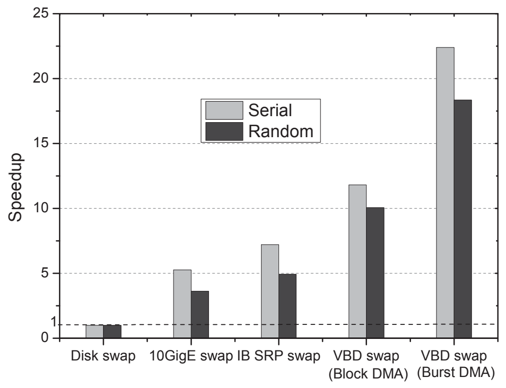


Figure 8: Micro-benchmark evaluations on Swap

4.1.3. DMA mode Two micro-benchmarks (serial access and random access) are used to evaluate the performance of using the remote memory as a swap space. Each node has 4GB memory and borrows 4GB memory from other nodes as its swap partition. Each benchmark uses a 6GB array. Since using the remote memory in this way can also be done through the Ethernet and IB, we compare five different implementations: local disk, 10GigE, 20Gb IB with SCSI RDMA Protocol (SRP), and VBD swap with block DMA, VBD with burst DMA). Figure 8 shows the speedup relative to the local disk swap. VBD with burst DMA has the best performance for both random accesses and serial accesses. It is almost 20 times faster than local swap.

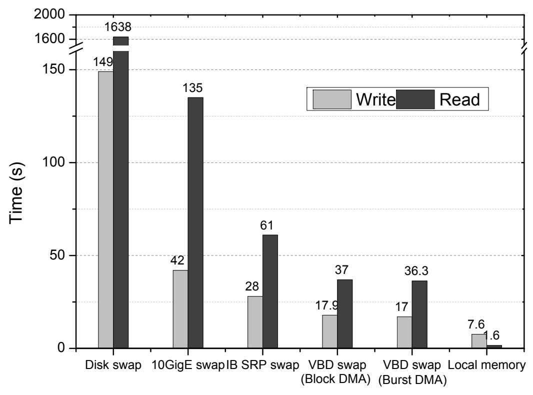


Figure 9: Performance evaluation for BerkeleyDB application

Another use of VBD is to store in-memory database in the remote memory. It totally performs 400000 sequential read operations followed by 400000 sequential write operations. Each operation accesses 1024 bytes, 4 bytes for the key and 1020 bytes for the data. It can be seen from Figure 9 that the prototype system outperforms others drastically. For this test, burst DMA and block DMA perform similarly because the test does not have burst access pattern that would benefit burst DMA. The experiment shows that using the remote memory for in-memory database is worth investigating. However, it still underperforms significantly relative to when the whole database is in the local memory.

Another way of using VBD is to make it behave like a local disk. Many traditional databasees require strong disk performance, so we use ORION (Oracle I/O Calibration Tool) [5] for this study. ORION is a standalone benchmark tool for calibrating the I/O performance of storage systems intended to be used for Oracle databases. In addition to the Ethernet, IB, and VBD implementations, we also measure two more alternatives, 7200rpm SATA disk and SATA3 SSD. Figure 10 shows the measured bandwidth, IOPS and latency. VBD once again stands out among the competitors. It has the highest bandwidth, highest IOPS, and lowest latency. Its bandwidth is 50 times higher than that of SATA disk, and 5 times higher than that of SSD. Its IOPS is 160 times higher than SATA disk and 3 times higher than SSD. Its latency is 600 times lower than SATA disk, and 7 times lower than SSD.

4.2. Sharing GPGPU

Four computing intensive workload kernels from the NVIDIA GPU Computing SDK [3] are chosen for this evaluation. Figure 11 shows the result. Using GPU remotely (RGPU) is almost as effective as using GPU locally for Matrix_Multiply and Optical_flow, achieving a speed of 185 times and 28 times respectively.

Using remote GPU requires additional memory copy and transfer operations not seen with using local GPU. The number at the tail of each arrow in Figure 11 is the percentage of the total execution time spent on transferring data to/from the remote node. For *separate convolution* and *fast Walsh transform*, the data transfer time dominates the execution time, leading to significant performance loss for RGPU. However, RGPU is still much better than just using local CPU.

4.3. Sharing NIC

Httperf toolset [1] is used to measure the bandwidth and latency for using remote 1GbE NIC. Figure 12 compares the bandwidth and latency between using a remote NIC (RNIC) and using a local NIC (LNIC). According to the figure, RNIC achieves more than 85 percent of the peak bandwidth of LNIC. Its latency is less than twice that of LNIC. Two factors contribute to the performance loss. First, RNIC needs additional memory copy. Second, the software switching consumes time. In addition, our VNIC implementation can copy data to remote hosts using DMA operations. Figure 12 shows that burst DMA is better than block DMA because VNIC can trigger burst DMA tasks for multiple packets at a time, leading to better bandwidth ultilization and less waiting time.

5. Discussions

From another angle, our work really attempts to find a design that offers many of the benefits of large-scale cc-NUMA systems but with a cost close to what clusters offer. The proposed prototype, while not perfect, does offer some nice benefits not seen with the cluster architecture. It utilizes the PCIe interfaces, which can be found in almost every mainline microprocessor. The PCI switch chips are also readily available. While it is hard to put a price tag on it now, it is our belief that such a chip could be made and purchased cost-effectively. We have discovered several technical features of the prototype system that warrant further investigation of such systems.

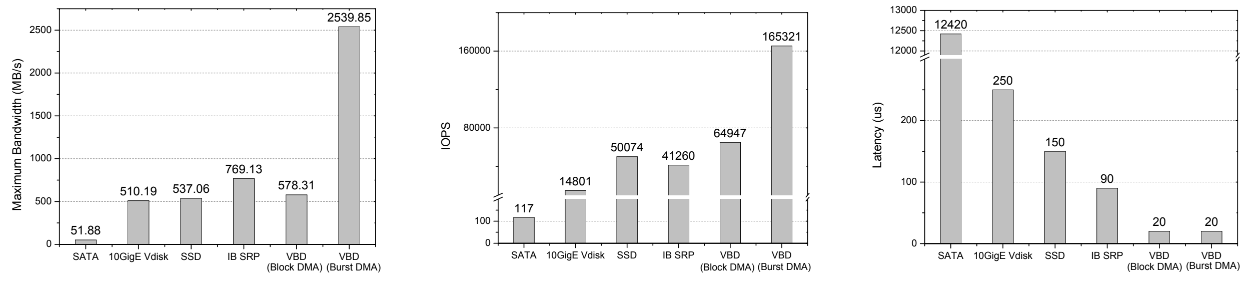
A PCIe switch based system can offer efficient memory sharing in a number of different ways. A node can use the memory located in another node as part of its memory, or as a swap space, or as a disk. A program can access the so-called remote memory through direct load/store, DMA requests, and block device requests. The efficiency of using the remote memory may be low for applications without good locality or with a high percentage of write operations. The inefficiency seems a direct result of some design choices in existing systems and can be addressed by further research.

Sharing computing capability, such as those offered by GPGPU, has huge potentials. While this would not be a problem if enough computing capability would be built into each node or a perfect scheduling algorithm would put every application on the right hardware, to the best of our experience, the reality is quite different. On the other hand, there is a lot of room for further improvement. For instance, allowing direct data transfer between the GPGPU memory and the remote memory would reduce the overhead of using remote GPGPU.

Sharing network bandwdith is practical as the performance degradation from using remote NIC to using local NIC is acceptable. Our implementation of IP over PCIe has the required functionality, but we believe further extension and/or optimization is possible. As a matter of fact, the whole communication software stack may be simplified or improved by means of software and hardware co-optimizations.

Our evaluation has mostly been using micro-benchmarks. It remains to be seen if real workloads will benefit from the proposed system. There is a lot of work remained to be done to fully exploit such systems. We remain confident that flexible resource sharing is a desriable feature of many data centers workloads. For instance, the large number of users and applications of a cloud computing center will have a wide spectrum of needs on every set of resource. Scheduling/grouping applications such that every application gets the exact amount of required resources would be made a lot easier on a system with the proposed resource sharing mechanisms.

PCIe is designed to connect local I/O devices with the CPU.



(a) Bandwidth (b) IOPS (c) Latency

Figure 10: Performance evaluation for Oracle ORION.

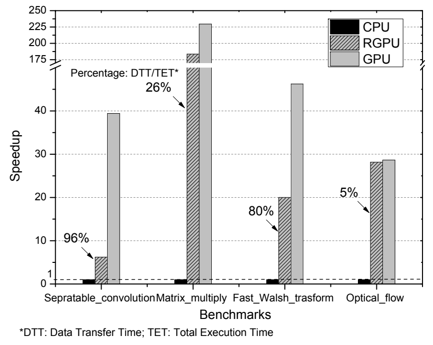
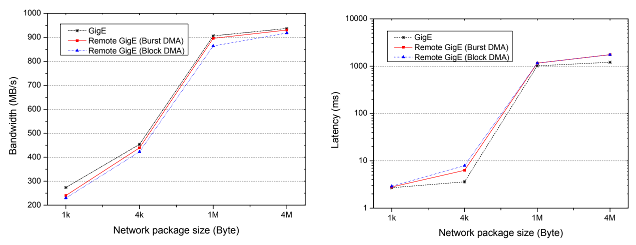


Figure 11: Performance evaluation for using remote GPU



(a) Bandwidth (b) Latency

Figure 12: Performance evaluation for using remote NIC.

The effective distance of a PCIe cable is normally 10 meters, far shorter than that of Ethernet and IB cables. This shortcoming makes it hard to build scale up a PCIe-based system. On the other hand, with the growing popularity of optical fiber interconnects, it is entirely possible to use optical cables between two PCIe interfaces by integrating optical transceivers on the board. With optical fibers, PCIe could scale up easily.

6. Related work

A few large internet companies, like Google, Amazon and Facebook, have successfully customized servers and data centers for their own set of applications [4, 9]. It has not been shown or made public if their customized systems works well with other types of applications.

Many efforts have been made to build cost-effective servers. The most common approach is to use non-server class components. Andersen et al. propose a cluster architecture that consists of a large number of slow, low-power embedded processors (AMD Geode LX or Intel Atom) coupled with Compact Flash or SATA-based Flash storage [7]. Kevin Lim et al propose a new server architecture for emerging warehouse-computing environments that incorporates embedded processors in novel packaging solutions with memory sharing and flash-based disk caching [17]. Malladi et al advocate server memory systems using mobile DRAM devices [19]. SeaMicro has built servers using low power processors (Atom or low-end Xeon processor) with proprietary interconnection technology [23].

Deshpande U et al have proposed the MemX system, which allows Xen VMs to transparently access the cluster-wide memory resources over Ethernet [11]. Similar mechanisms can be found in other studies [24, 16, 8]. In [18], blades with a large capacity of memory are introduced as way to expand memory capacity. J. Ousterhout et al. propose RAMClouds that puts the entire data set into the aggregated memory from thousands of commodity servers [20].

Byrne et al have demonstrated that PCIe switch in a Non-Transparent Bridge mode can provide higher bandwidth and lower latency over 1Gb Ethernet when running hadoop-based applications [10]. Leigh et al. have studied the cost/bandwidth advantage of PCIe over Ethernet [15]. Venkata Krishnan introduced the IPoverPCIe [13] and implemented the bypass kernel version [14]. These work provides plenty of proof that PCIe has the potential to be a data center interconnect technology. Beyond that, our work identified the performance bottlenecks via in-depth system-level performance evaluations, and proposed three kinds of resource sharing mechanisms to build cost-effective data center servers.

7. Conclusion

The number of applications that run on data center servers is growing rapidly. The conventional cluster architecture is failing to meet the needs of many of the new applications. A few mega companies like Google are customizing servers for their own applications. Many IT service providers are in dire need of new servers that can support their applications/services efficiently while keeping the cost of their data centers under control.

The paper advocates a system where computer nodes are connected through the PCIe interface, an interface that is initially designed to connnect local peripheral components to the host processor. We implement various mechanisms to allow resources of each node to be shared by other nodes in the system. With these mechanisms and proper software stacks, one node can dynamically use the idle resources of other nodes as they were its own local resources.

We have built a prototype system and studied the benefits and issues of sharing memory, GPGPU, and network bandwidth on the system. Our findings are as follows.

(1) Memory can be shared in multiple different ways. Any sharing through DMA requests performs wonderfully. Sharing through direct loads/stores may incur high overhead with the modern processors that contain no special optimization for cache line loads/stores through the otherwise considered pure I/O interface.

(2) Sharing I/O devices such as GPGPU and NIC is both feasible and beneficial. It allows resources within a system to be better utilized. For instance, a node running a network intensive application and a node running a computing intensive application can exchange the GPGPU of the first node with the NIC of the second node.

(3) The prototype system could use extensions and improvements in multiple areas. The key challenge is again to come up with practical solutions without increasing the cost. Multiple issues exposed by the prototype could be fixed by changing the processor. However, any change to the processor chip is no easy task.

Our study deviates from typical research work that seeks the best performance or the best power efficiency. Instead, it is about finding the most cost effective solutions that can have a widespread impact. What we have found is that this angle of research is also full of challenges that require clever innovations. We hope our paper will be able to provide some insights and allude to enough interesting topics for those who might be interested in this type of research work.

8. Acknowledgements

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under grant No. XDA06010401, and National Science Foundation of China under grant No. 61100010. We would like to thank Lixin Zhang for his help in idea formulation and paper writing. We also wish to thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] HP HTTPerf Benchmark. http://www.hpl.hp.com/research/linux/httperf/.

[2] InfiniBand Performance. http://www.mellanox.com/content/pages.php?pg=performance_infiniband.

[3] NVIDIA GPU Computing SDK.. http://developer.nvidia.com/cuda/gpu-computing-sdk.

[4] Open Compute Project. http://opencompute.org/.

[5] Oracle ORION benchmark.. http://www.oracle.com/technetwork/topics/index-089595.html.

[6] Tektronix PCIe Logic Protocol Analyzer. http://www.tek.com/datasheet/protocol-analyzer/tla7sa00-tektronix-pci-express-logic-protocol-analyzer.

[7] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: a fast array of wimpy nodes," *ACM Commun.*, vol. 54, no. 7, pp. 101–109, Jul. 2011.

[8] E. Anderson and J. Neefe, "An Exploration of Network RAM," UC Berkley, Tech. Rep. CSD-98-1000, Aug. 1998.

[9] L. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22 – 28, Mar-Apr 2003.

[10] J. Byrne, J. Chang, K. T. Lim, L. Ramirez, and P. Ranganathan, "Power-efficient networking for balanced system designs: early experiences with PCIe," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, USA: ACM, 2011, pp. 3:1–3:5.

[11] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan, "MemX: Virtualization of Cluster-Wide Memory," in *The 39th International Conference on Parallel Processing (ICPP)*, Sep. 2010, pp. 663 –672.

[12] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[13] V. Krishnan, "Towards an integrated IO and clustering solution using PCI express," in *IEEE International Conference on Cluster Computing*, Sep. 2007, pp. 259 –266.

[14] V. Krishnan, "Evaluation of an Integrated PCI Express IO Expansion and Clustering Fabric," in *The 16th IEEE Symposium on High Performance Interconnects (HOTI '08).*, Aug. 2008, pp. 93 –100.

[15] K. Leigh, P. Ranganathan, and J. Subhlok, "Fabric convergence implications on systems architecture," in *The 14th IEEE International Symposium on High Performance Computer Architecture (HPCA 2008).*, Feb. 2008, pp. 15 –26.

[16] S. Liang, R. Noronha, and D. Panda, "Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device," in *IEEE International Cluster Computing*, sept 2005, pp. 1–10.

[17] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *the 35th International Symposium on Computer Architecture (ISCA '08)*, Jun. 2008, pp. 315 –326.

[18] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, USA: ACM, 2009, pp. 267–278.

[19] K. Malladi, F. Nothaft, K. Periyathambi, B. Lee, C. Kozyrakis, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *The 39th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2012, pp. 37 –48.

[20] J. Ousterhout *et al.*, "The case for RAMClouds: scalable high-performance storage entirely in DRAM," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010.

[21] PLX. ExpressLane PEX 8619-BA 16-Lane, 16-Port PCI Express Gen 2 Switch with DMA Data Book. http://www.plxtech.com/products/expresslane/pex8619.

[22] PLX. ExpressLane PEX 8648-AA AB, and BB 48-Lane/12-Port PCI Express Gen 2 Switch Data Book. http://www.plxtech.com/products/expresslane/pex8648.

[23] A. Rao, "SeaMicro Technology Overview." Available: http://www.seamicro.com/sites/default/files/SM_TO01_64_v1%208.pdf

[24] A. Samih, R. Wang, C. Maciocco, T.-Y. C. Tai, and Y. Solihin, "A collaborative memory system for high-performance and cost-effective clustered architectures," in *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*, ser. ASBD '11. New York, USA: ACM, 2011, pp. 4–12.

[25] W. Zhang. PEX 8619 DMA Performance Metrics. http://www.plxtech.com/products/expresslane/pex8619.