Characterization of Input/Output Bandwidth Performance Models in NUMA Architecture for Data Intensive Applications

Tan Li, Yufei Ren, Dantong Yu*, Shudong Jin, Thomas Robertazzi
Department of Electrical and Computer Engineering,
Stony Brook University, Stony Brook, NY, 11794, USA
Email: tan.li@stonybrook.edu, yufei.ren@stonybrook.edu, shudong.jin@stonybrook.edu, thomas.robertazzi@stonybrook.edu
*Computational Science Center,
Brookhaven National Laboratory, Upton, NY 11973, USA
Email: dtyu@bnl.gov

***Abstract*—Data-intensive applications frequently rely on multicore computer systems, in which Non-Uniform Memory Access (NUMA) is a dominant architecture. To transfer data into and out from these high-performance computers becomes a bottleneck, and thus it is crucial to understand their I/O performance characteristics. However, the complexity in NUMA architecture presents a new challenge in modeling its I/O access cost, and thus lead to difficulties in configuring proper processor and memory affinity. In this paper, we show that existing NUMA experimental methods and metrics are inappropriate on contemporary high-end systems. We characterize a state-of-the-art NUMA host, and propose, to the best of our knowledge, the first methodology to simulate I/O operations using memory semantics, and model the I/O bandwidth performance. Our methodology is thoroughly tested and validated by mapping multiple parallel I/O streams to different sets of hardware components (CPU, memory, network cards, and SSDs) and by measuring the performance of each mapping. The experimental results and analysis reveal that our methodology can dramatically reduce characterization workload, accurately estimate the overall I/O performance, and effectively mitigate resource contention among I/O tasks.**

***Index Terms*—Data Transfer, Input/Output(I/O), NUMA effects, Performance model**

I. Introduction

Many current high-end systems contain multiple CPU packages (sockets) on a single motherboard. Processors, memory banks, and Input/Output (I/O) modules that are distributed across different domains (nodes) and assembled together by a cache-coherent, point-to-point interconnect. Each processor has faster access to the directly-attached memory modules than the remote ones linked by these interconnect buses. Such a feature is widely known as Non-Uniform Memory Access (NUMA). This performance asymmetry becomes more severe with the current trend of putting more CPUs and cores into a single host at the expense of losing uniformity [1]. As shown in Table I, a host with more nodes will be more severely affected by the NUMA effect due to longer average latency than the smaller one [2]. Here the NUMA factor is defined as the ratio between remote access latency versus local one.

TABLE I
NUMA FACTOR OF DIFFERENT SERVER CONFIGURATIONS

Server type	NUMA factor
Intel 4 sockets/4 nodes	1.5
AMD 4 sockets/8 nodes	2.7
AMD 8 sockets/8 nodes	2.8
HP blade system 32 nodes	5.5

However, the term "NUMA" is misleading since memory is not the only resource affected by the asymmetric nature of a NUMA architecture. Previous studies exposed a significant I/O performance variation in both latency and bandwidth. When an I/O access is remote, latency increases and maximum bandwidth usually decreases for data transfer. For example, one research [3] benchmarked TCP performance on two different NUMA testbeds. Their measurement showed that the placement of the process on remote CPU cores, at either sender or receiver side, can lead to as much as a 30% loss of the overall TCP bandwidth performance. In [4], the bandwidth and latency performance of PCIe-attached GPU hardware was tested by various benchmark tools to compare the performance of shared I/O hub and dedicated I/O hub in NUMA systems. The results showed that the penalty of incorrect NUMA assignment is substantial and asymmetric. A 31% reduction in readback bandwidth and a 13% reduction in download bandwidth were observed for bulk data transfer to GPU devices.

On the other hand, a recent study [5] showed that, for I/O devices with higher maximum bandwidth, a larger performance drop is observed if the placement is not aligned with local memory or CPU. With the introduction of a new generation of high performance network adapters such as $40Gbps$ Ethernet and $100Gbps$ InfiniBand server adapters [6], and Solid-State Drives (SSD) such as the LSI Nytro Drive [7], the performance problem deteriorates even more. This further warrants a rethinking of memory and I/O interface affinities for data-intensive task scheduling.

A. Motivations

Recent studies showed that maximizing data locality does not always minimize the execution time of data intensive applications, especially in a multi-user/multi-task cluster environment. An accurate NUMA cost model is more suitable to represent every detailed aspect of the system performance features. The work in [8] indicated that a fraction of the node resources are specially reserved for remote memory accesses on the Intel Nehalem and Westmere platforms that take fairness into consideration. Hence, allocating all the tasks locally cannot take full advantage of the aggregate system memory bandwidth. Moreover, excessive use of local resources will introduce contention and congestion among concurrent tasks on shared queues and buses, and thus degrade the overall system performance [9].

Most of current performance models and resource assignment algorithms for NUMA architecture [10], [11], [12] are based on hop-distance, directly or indirectly. Hop-distance is one of the most popular NUMA metrics, and represents the number of physical links along the data access path between two devices. More hops on this path usually imply a higher access cost. However, hop-distance is not a good indicator of NUMA penalty, especially to I/O performance, due to the following reasons.

First, the architectural details of many modern NUMA systems often are not intuitive to users. In [13], the AMD architecture designers illustrated three possible topologies of the 4P AMD Opteron Magny Cours platform, as shown in Figure 1(a,b,c). For the same type of 4P processors, the authors of [3] described another topology variant as Figure 1(d). The exact design of a NUMA system depends on the choices made by system architects, and is therefore implementation-specific, even for the same technology specification.

Second, even if the full topological details are known, hop-distance is still not an accurate metric for actual data transfers among CPU, memory, and I/O modules. The authors of [1] demonstrated, for their eight-node AMD host, an overhead due to cache coherency traffic led to a higher penalty to cores on the edges compared to middle ones in the host topology. Even local memory access can have a significant performance difference over CPU nodes. Moreover, the study in [14] also showed the impact of OS noise on application performance in NUMA systems.

Third, it is less straightforward to measure I/O bandwidth than memory bandwidth, since the bandwidth of peripherals depends on all resources along the data path between I/O devices and processors. These resources include PCIe lanes, CPU interconnect links, various controllers and chipsets, and the specific PCIe devices that contain the requested data. The performance bottleneck can reside in any of these. Furthermore, I/O bandwidth is often an order of magnitude lower than memory bandwidth, and the I/O performance difference among various NUMA configurations is not as obvious as the memory access cases.

Consequently, the only way to understand memory and I/O device access characteristics is to test and analyze all possible access scenarios using empirical methods. However, as we will present later, existing NUMA-related tools and experimental approaches cannot reveal the precise attributes of I/O bandwidth performance on a given architecture. This motivates our work presented in this paper.

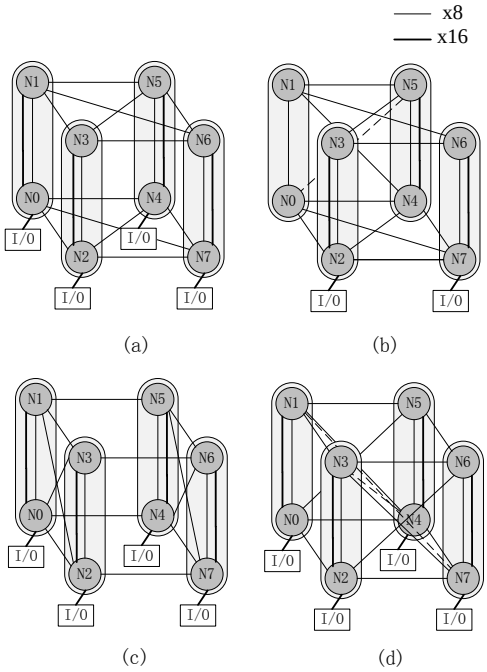


Fig. 1. Possible topologies of 4P AMD Opteron Magny Cours Processors. The oval circles in the plot represent the NUMA nodes, including both CPU and memory. The lines between them denote high-speed HyperTransport (HT) interconnections. The width of a link can be 8 or 16 bits.

B. Our Contributions

In this study, a comprehensive memory and I/O bandwidth performance characterization is described, using the state-of-the-art NUMA servers with high-performance network adapters and SSDs. First, we show the limitations of existing characterization methods, including the ineffectiveness of using hop-distance to measure the NUMA penalty, and the limitations of the well-known STREAM benchmark [15] to analyze I/O performance. Second, we propose an empirical method to characterize and predict the relative bandwidth performance levels among various NUMA configurations without involving the actual I/O devices. The method is validated by benchmarking I/O operations, including TCP, Remote Direct Memory Access(RDMA) and disk read/write, in a node level. Third, we design and develop, to the best of our knowledge, the first NUMA characterization software for bulk data I/O tasks. We claim that, with our NUMA characterization method and tool, accurate I/O bandwidth performance models can be obtained, and used to improve application I/O behavior and to assist runtime schedulers for all NUMA platforms.

The remainder of this paper is organized as follows. Section II explains the terminologies and existing OS support for NUMA optimizations. The detailed configurations of the

NUMA host for evaluation and the benchmark tools are presented in Section III. In Section IV, we describe the experimental results and the analysis of memory and I/O bandwidth performance characteristics. Section V describes the design of our proposed empirical methodology and the corresponding benchmark software. The conclusions and future directions are summarized at the end.

II. Technical Background

A. Terminology

As NUMA systems scale up, it is prohibitively difficult to implement a full connection for all processors in a host due to hardware constraints. For instance, the pin constraint of the AMD G34 (Generation 3, four memory channels) architecture allows at most four HyperTransport (HT) ports per CPU node. Additionally, for the bottom nodes in the topology, as shown in Figure 1, one port is reserved for I/O peripherals. These design constraints prohibit a fully-connected topology, and create different physical distances for remote accesses. In Figure 1, the two CPU dies in the ellipses are physically put in a tightly coupled multi-chip CPU package. We define "local" node as all the resources, including CPU cores, memory banks, I/O devices, directly attached to a single CPU die, and a "neighbor" node as the resources that are not local, but in the same CPU package, and "remote" nodes as the resources in other packages. For example, as shown in Figure 1(a), node 7 is local to itself, a neighbor to node 6, remote to nodes $\{0, 2, 4\}$ with one hop, and to nodes $\{1, 3, 5\}$ with two hops. This implies multiple possible performance levels to access data.

B. NUMA support in Linux

Modern operating systems are designed to be NUMA-aware. For example, the default memory policy in Linux kernel 2.6 is *local preferred*. It means that an application's data is allocated in its local NUMA node if there is sufficient memory space available. However, the access pattern to I/O peripherals depends on individual applications. It is possible for an application to request an I/O device that is remote to the processing CPU core and memory regions, and therefore the application will suffer from remote access penalty.

To support the user level control of NUMA configurations, Linux system provides a collection of libraries, command line tools, and hardware counters as follows [16], [17].

- **numastat** displays the NUMA memory allocation statistics, including the number of hit and miss events of memory page allocations, from kernel memory allocator.
- **numademo** is a benchmark which shows the effect of possible resource affinity policies, such as local, remote, and interleave. It includes seven test modules, such as memset, memcpy, and also the STREAM benchmark.
- **numactl** is a command line utility to configure a specific NUMA schedule policy for an executable task and all its child processes. It also can provide the basic system NUMA information, such as the relevant distance between two nodes, but this distance is often inaccurate [18].
- **libnuma** is a shared library which allows applications to issue system calls to dynamically configure their own NUMA policy at runtime.

Several other tools provide additional system information regarding the NUMA settings. For example, the Portable Hardware Locality (hwloc) utility [19] can be used to analyze /proc and /sys file systems in Linux and provide a systemic view of a host architecture. However, it does not include the information regarding how the NUMA nodes are interconnected in a system.

C. DMA engine for I/O and memory operations

DMA (Direct-Memory Access) refers to the I/O strategy where I/O hardware is allowed to directly read and write memory without involving CPU. Many modern high-performance devices, such as SSDs, network adapters, and graphic cards, have their built-in processors and controllers for DMA. DMA can also be used for "memory-to-memory" copying within host memory to offloading the tasking of copying a large block of data copy from CPU. In this paper, we exploit bulk data transfers within host memory to simulate the real DMA transfer between host memory and PCIe devices, and try to obtain the NUMA characteristics of I/O device access.

III. System Configurations for Characterization

In this section, we describe the testbed hardware and the software benchmark configurations in our experiments.

A. Server hardware specifications

Table II lists the system model and configurations of our NUMA systems. Each system has four AMD processors and each processor contains two CPU dies, and thereby two NUMA nodes. An I/O hub is attached to one of the two dies in the package. The CPU sockets and I/O hubs are linked by HT 3.0 interconnections [20], and their topology can be any of those in Figure 1.

The target host is equipped with two LSI SSDs and a 40Gbps network adapter with the capability of RDMA over Converged Ethernet (RoCE) [21]. Another identical host is used in the network performance test, as shown in Figure 2. All network adapters and SSDs are directly attached to node 7, and we will use node 7 as the exemplary node hereafter. All network interfaces are optimized based on vendor's recommendations [22]. The Round Trip Time (RTT) between the two hosts is about 0.005ms, as reported by *ping*.

B. Benchmarks and affinity settings

The focus of our work is the relative user-level bandwidth performance under all possible NUMA scenarios. As stated in [5], improving the absolute bandwidth can lead to more noticeable NUMA effect, and thus facilitate our comparison and analysis. Therefore, we optimize our testbed and benchmark settings as follows.

TABLE II
CONFIGURATION OF THE AMD 4P SERVER

Motherboard	HP ProLiant DL585 Gen 7
Chipset	AMD SR5690/SP5100
CPU Model	AMD Opteron 6136 Magny-Cours @ 2.4GHz
CPU cores/NUMA nodes	32/8
Memory	32GB
Last level cache(LLC)	5MBytes
I/O Bus	PCI Express Gen 2 x8 lanes
Linux Kernel	2.6.32-279.19.1.el6.x86_64
SSD Drive	LSI Nytro WarpDrive WLP4-200 Card
Network Interface Cards	ConnectX-3 EN Dual Port 40 Gigabit Ethernet Adapter
Network Interface Card Driver	MLNX_OFED_LINUX-1.5.3

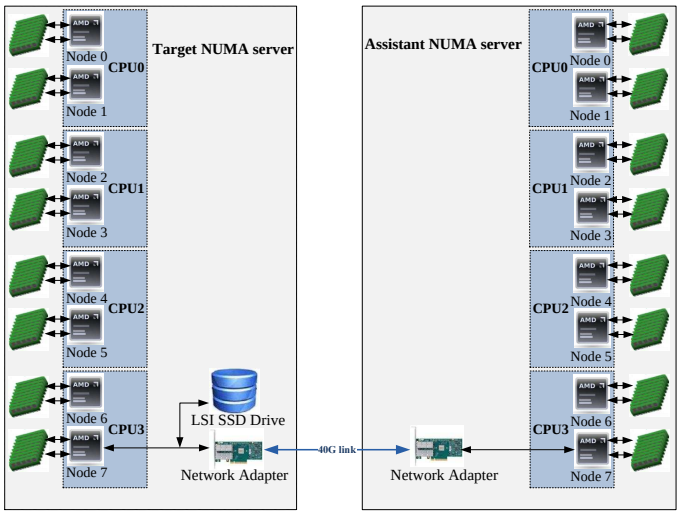


Fig. 2. System connection diagram for characterization. All PCIe devices are connected to node 7.

1) Memory benchmark: The STREAM benchmark is used to determine the maximum aggregate memory bandwidth between two NUMA nodes. The performance measurements reported by STREAM highly depend on compilers. We optimize the benchmark compilation according to the recommendation of the AMD technical document [23], and also take advantage of the *-fopenmp* compiling flag to enable multi-threaded tests. The STREAM benchmark executes four types of memory access operations on large data arrays, but they exhibit a similar performance on modern machines. Herein, we choose the *Copy* operation for our characterization, because it contains no computation, and is similar to I/O data transfer behavior. In order to eliminate CPU cache effect, STREAM requires that each array be at least four times as big as the largest cache used. In our case, the LLC size is $5MBytes$ per CPU die, and thereby the array contains at least $20MBytes$, or $2,621,440$ long integers.

2) I/O benchmark: Flexible I/O Tester (fio) [24] is a user-level benchmark tool used to characterize system's PCIe device access performance. This test tool spawns a number of processes performing I/O jobs with user-defined I/O operations and desired parameters. It supports a wide spectrum of I/O operations, such as disk read/write and TCP/UDP network data transfers. We also added RDMA engines into this tool, and extended its capability to support RDMA_READ, RDMA_WRITE, and SEND/RECEIVE operations [25]. Furthermore, we redirected the hardware interrupts generated by I/O devices to their local CPU node.

IV. EXPERIMENTAL CHARACTERIZATION

A. Memory performance characterization

In this section, for each test case, the STREAM benchmark is set to run 100 times, and it reports the maximum observed bandwidth instead of the mean value. $numactl$ is used to pin each benchmark process/thread to a desired CPU/memory node. Benchmarking every memory data access scenario in a NUMA system entails mapping benchmark threads and their memory in every possible configuration that a data intensive program might encounter during its execution. For modern multi-core systems, mapping threads to individual cores greatly expands the test set. Cores attached to the same NUMA node are supposed to show the identical memory and I/O bandwidth when accessing data on a given node, as shown in previous literature [3], [18]. Hence, we need only to focus on node-level characterization. Meanwhile, four parallel threads are used in each test, since each NUMA node has four CPU cores.

After benchmarking, a $N \times N$ bandwidth matrix can be obtained, where N is the number of configured NUMA nodes in a host. This is shown in Figure 3. The bandwidth performance exhibits asymmetry among tests. For example, when STREAM runs on node 7 to access data from node 4, we obtain a bandwidth of $21.34Gbps$, which is better than the two cases of accessing data on node $2, 3 respectively$. However, when the benchmark runs on node 4 to access data on node 7, only $18.45Gbps$ is achieved, and this is worse than the performance of running the benchmark on node $2, 3$. This issue can result from the asymmetric setup in the AMD hardware, such as the number of request and response buffers, and link width configuration for cache coherent traffic [20], [26].

Another noticeable observation is that when STREAM runs on node 0 to access the data on the same node, the performance is apparently higher than that of other local NUMA binding cases. Note that all OS buffers and shared libraries reside on memory allocated in node 0, and thus the cores of node 0 have an unfair advantage compared to the other processors. To validate our observation, we use the "*numactl –-hardware*" command to show the size of free memory per node. Executing this command without any application running, we notice that all nodes have almost $4GBytes$ free memory, except for the first one with only $1.5GBytes$ free memory. This observation proves that the first node uses more memory than the others in an idle system.

Furthermore, from the test plots, we can see the local node has the best performance, and the neighboring node has the second best. If we use hop-distance as the decisive factor of the NUMA cost, then we should have observed: 1) the nodes

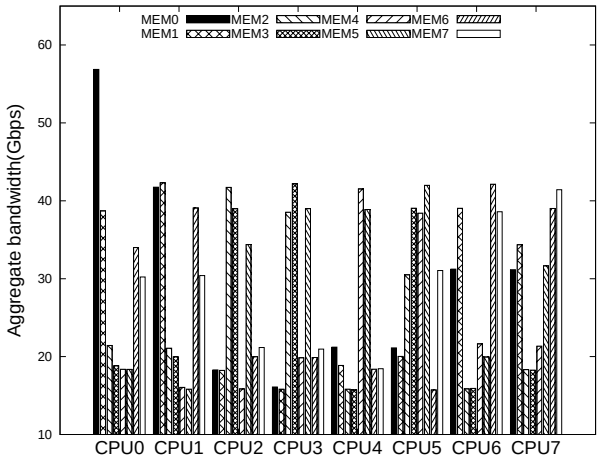


Fig. 3. Bandwidth performance model by STREAM benchmark copy operation. "CPUn" denotes all STREAM test threads running on node n, and "MEMn" denotes all test STREAM threads are accessing the data on node n.

with the highest bandwidth are local nodes, 2) the second best one is the nodes that are one hop away, and, 3) the nodes with two hops aways have the lowest bandwidth. We can then derive the topology of our tested NUMA host. However, the connectivity inferred from the test data does not match any of the topologies shown in Figure 1. We cannot either draw any reasonable topology due to the performance asymmetry we just mentioned. Therefore, it is inappropriate to simply use the physical distance to determine the NUMA cost for memory bandwidth performance modeling.

B. I/O performance characterization and analysis

Since hop distance cannot represent the accurate memory bandwidth performance model, we resort to benchmark approaches. One benchmark is STREAM. For example, in [18], the authors built a memory access cost model via the STREAM benchmark, and then confirmed it with multiple other benchmark tools. They then integrated the model into a benchmark toolkit called *cbench* [27]. In this section we will examine this approach as well.

We provide a full characterization of node 7 using STREAM benchmark and the same configuration in Section IV-A, and show the results in Figure 4. Figure 4(a) depicts the cases when STREAM benchmark runs on node 7 to access data on all nodes. We call this model a "CPU centric" characterization. Figure 4(b) illustrates the cases when the data is on node 7 and is accessed by all nodes. This is called "memory centric".

1) TCP performance characterization: Here we present the analysis of high-speed network data transfer across the PCIe interconnect to/from a selected NUMA node with both TCP and RDMA protocols. This involves measuring write rate (sending data to I/O devices) and read rate (receiving data from I/O devices). As stated in Section II-B, the default NUMA policy in current Linux is *local-preferred*. The I/O applications can always be guaranteed with local memory space if possible. However, application threads may run on a node that is remote

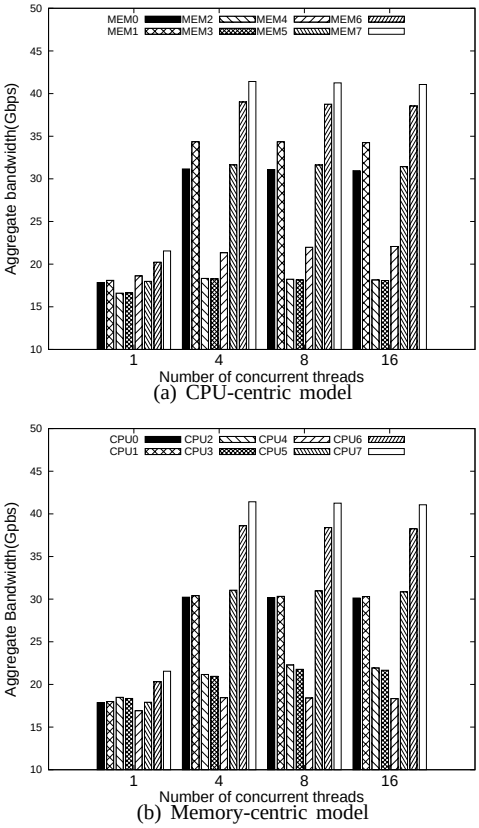


Fig. 4. Bandwidth performance models of node 7, produced by STREAM benchmark

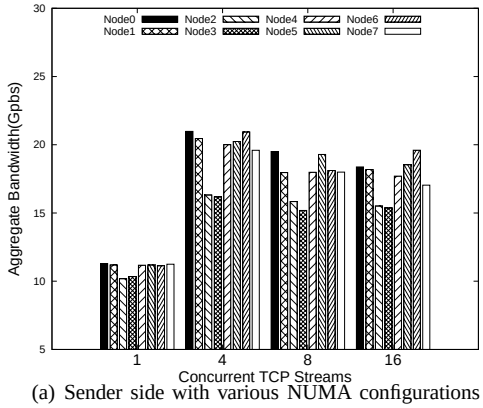
TABLE III
PARAMETERS FOR NETWORK I/O TESTS, INCLUDING TCP AND RDMA

Data size requested by each test process	400GBytes
TCP Variant	Cubic
IO block size	128KBytes
Ethernet frame size	9000

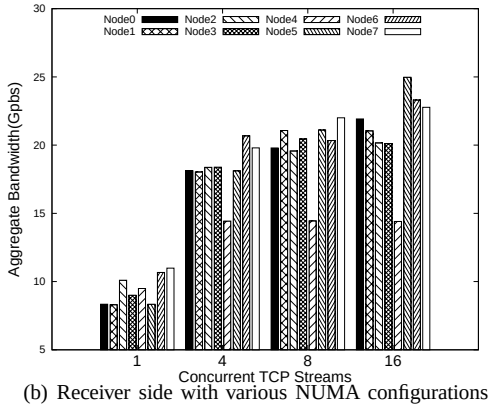
from I/O devices. The application performance model can act as either CPU centric or memory centric, as shown in Figure 4.

Table III describes the network test configurations. In order to get an accurate and stable bandwidth performance, each data stream is required to transfer $400GBytes$ of data, and the average aggregate performance is then reported. All test cases will allocate buffers in their local memory space, and then vary the NUMA node where the fio benchmark is executed. Figure 5 shows the aggregate bandwidth performance with various numbers of concurrent streams, and with different NUMA setups on data sender side and receiver side, respectively.

The network adapters use the PCIe Gen 2.0 8x peripheral interface which supports a maximum of $40Gbps$ raw transfer rate. Due to the 8/10bit encoding used for the PCIe Gen2 protocol, the available data bandwidth is degraded to $32Gbps$. The real available bandwidth is further decreased by the inherent overhead of network protocols (Ethernet, TCP/IP, RDMA). Therefore, the maximum bandwidth of $25Gbps$ in our tests is very close to the theoretical performance limit.

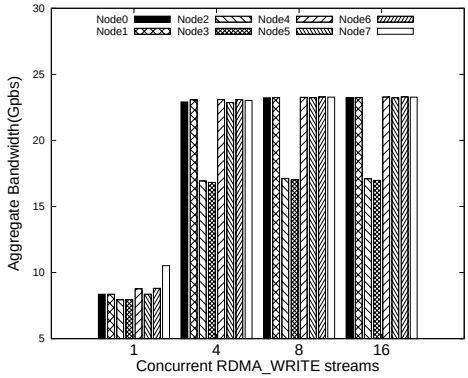


(a) Sender side with various NUMA configurations

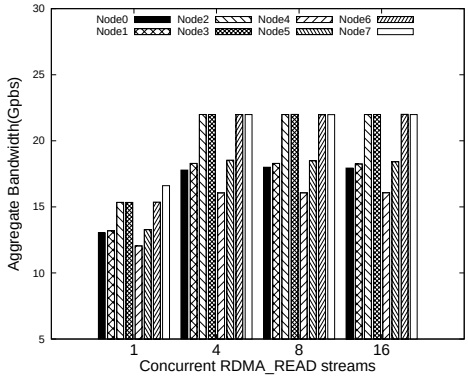


(b) Receiver side with various NUMA configurations

Fig. 5. TCP bandwidth performance characteristics



(a) RDMA_WRITE with various NUMA configurations



(b) RDMA_READ with various NUMA configurations

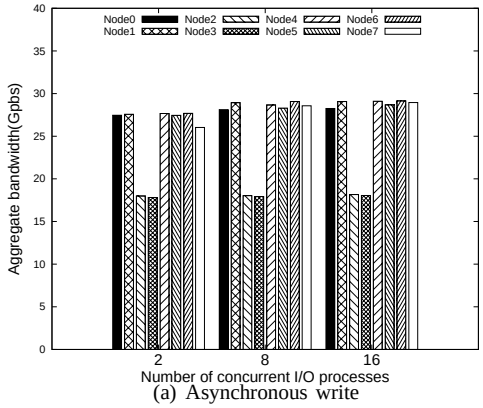
Fig. 6. Bandwidth performance characteristics of RDMA operations

For TCP performance in Figure 5, the bandwidth grows when the number of concurrent streams increases until there are four parallel streams. When the number of concurrent streams further rises, the contention among them begins to introduce some unexpected behavior. Therefore, sometimes, the performance of node 5 appears to be the best when there are eight or sixteen current threads. Overall, it seems that the TCP send performance in Figure 5(a) is close to that in the CPU centric model, while the TCP receive performance in Figure 5(b) is close to that in the memory centric model.

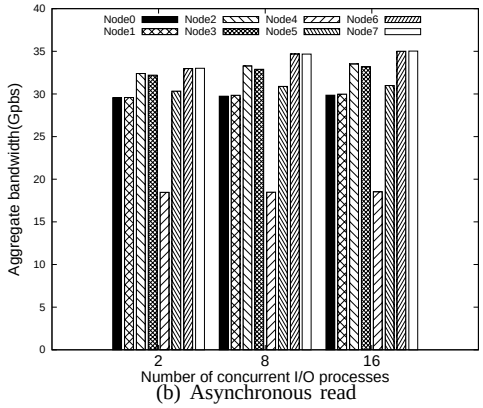
Another finding is, when the data path is bound to the local node 7, the performance is not always the best. In many cases, we can get better performance when we allocate CPU and memory resources on node 6, the neighboring node of node 7. This is because all interrupts from I/O device are handled locally by node 7, as stated in Section III-B2. When we run all application processes on node 7, the contention among multiple tasks will lead to a performance degradation. On the other hand, node 6 also has its own on-chip access to the I/O devices, and does not have the distraction of I/O interrupt handling, and therefore this binding results in an even better performance than the local node 7.

2) RDMA performance characterization: For the RDMA performance in Figure 6, the bandwidth is more stable than that of TCP. That is because RDMA operations offload most of their protocol precessing to network adapters, and significantly reduce resource contention on host. RDMA_WRITE bandwidth performance is close to that in the CPU centric model, but RDMA_READ does not match with neither the CPU centric model nor memory centric model in Figure 3. For example, in both models of Figure 3, the performance of node {0, 1} cases is better than that of node {2, 3} cases by 43% to 88%, respectively. When RDMA_READ runs on node {0, 1}, its bandwidth performance is worse than that of node {2, 3} by 15% to 18.4%. By looking at the TCP receive performance in Figure 5 again, we can see that the bandwidth of node {2, 3} cases still slightly outperforms that of node {0, 1} cases. All above indicate that the performance models in Figure 3 cannot be applied to network I/O traffic.

3) Disk I/O performance characterization: In order to further confirm the mismatching of performance levels we found in Section IV-B2, we provide the bandwidth characterization for PCIe-based SSDs using the fio benchmark. We observe that regular kernel-buffered read/write operations perform much worse than kernel-bypassed ones, and asynchronous I/O operations outperform synchronous ones on our testbed. Therefore, we utilize the libaio engine with the kernel-bypass option to maximize transfer speed. In order to further increase the aggregate bandwidth, two LSI SSD cards are accessed simultaneously, and the overall bandwidth performance is reported. Hence, the total number of test processes is at least two. Each test process transfers $400GBytes$ of data, with a



(a) Asynchronous write



(b) Asynchronous read

Fig. 7. Bandwidth performance characteristics of disk I/O operations

block size of $128KBytes$ and an I/O depth of 16. Figure 7 depicts the aggregate bandwidth performance with multiple test processes. As expected, the disk write rate corresponds to the TCP/RDMA send rate in previous tests, and the disk read rate here corresponds to the TCP/RDMA receive rate. However, none of these I/O performance characteristics is consistent with the STREAM benchmark results in Figure 3.

C. Analysis of performance mismatching

All above results proved that the NUMA characteristics captured by STREAM benchmark do not match with the I/O performance model, especially when application processes read data from I/O devices. The performance inconsistency among the models produced by STREAM and I/O operations may be ascribed to the following reasons:

- **Transfer data using PIO engine or DMA engine.** In STREAM test, an application requests a large amount of memory to create and initialize two big arrays of $double$ values, and then copies one array to the other, one item at a time. For this small message transfer, CPU is supposed to utilize Programed I/O (PIO) to move the data by itself. On the other hand, for bulk data transfer between memory and I/O device, CPU will offload the I/O task to DMA engine, and the actual data transfer will therefore bypass CPU. According to the AMD specifications [26], the difference of routing methods between CPU core access and memory access makes us believe that the PIO communication and DMA communication can have distinct paths within the involved hardware.
- **Data source and sink locations of data transfer operations.** The STREAM benchmark itself does not support NUMA awareness for either the location of program code or the location of allocated memory. To enable it to be aware of the NUMA architecture, we rely on the *numactl* command line tool to statically binding all the memory and CPU resources during the entire execution time. In *Copy* operation, both the data source and sink can only be bound to the same NUMA node, as illustrated in Figure 8(a). Meanwhile, for I/O operations, the source and sink can be in different NUMA nodes, as shown in Figure 8(b). This difference can also lead to different performance characteristics.

V. NUMA CHARACTERIZATION METHODOLOGY FOR I/O OPERATIONS

In the previous section, we showed that hop-distance and NUMA characterization based on STREAM benchmark cannot capture I/O performance attributes. Therefore, we need other appropriate methods, and in this section, we propose a new one.

A. Proposed methodology for NUMA I/O performance model

We utilize *memcpy* operation and *libnuma* library to move a large bulk of data between a given source memory node and a target memory node, as shown in Figure 8(c). Herein, a target NUMA node refers to the node that is directly attached to I/O devices and needs to be characterized. In this way, we assume that a CPU offloads slow data copy operation to the device's DMA engine. We can simulate the behavior of the DMA engine with a "memcpy" process that is bound to the target NUMA node. The operation of our proposed methodology solves the two mismatched behaviors we mentioned in Section IV-C, and therefore can be utilized to build a bandwidth performance model for PCIe based I/O tasks. The detailed procedure of our proposed methodology is shown in Algorithm 1. It first finds out the number of parallel threads, i.e., the number of CPU cores in a single NUMA node. Then multiple test threads are initiated to copy data simultaneously and independently.

For PCIe based I/O operations, CPU offloads data transfer tasks to the DMA engine in I/O devices. After this, DMA will take care of data transfers. For data write cases, the DMA engine reads data from the host memory, and stores it into the buffers inside I/O devices. In data read cases, DMA reads data from the buffers in the I/O hardware, and writes to the host memory. To simulate these scenarios, we enforce all the data-copy threads running on the target node to simulate a DMA engine in I/O devices. In the simulating cases of writing into I/O devices, the data sink is statically bound to the target node and the source node varies among the tests, as shown in Figure 9(a). In the reading test cases, we fix the source to the target node and vary the data sink node, as illustrated

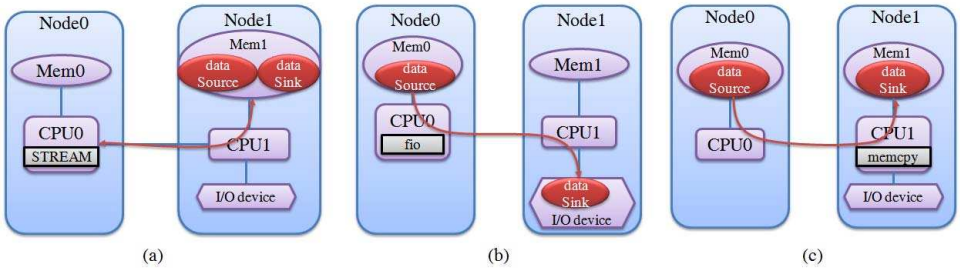


Fig. 8. (a) Data copy process in STREAM benchmark. Both data source and sink are located in node 1. (b) Data copy process in I/O operations. Data source is in node 0 and sink is in node 1. (c) Data copy process in proposed method. Data source is in node 0 and sink is in node 1.

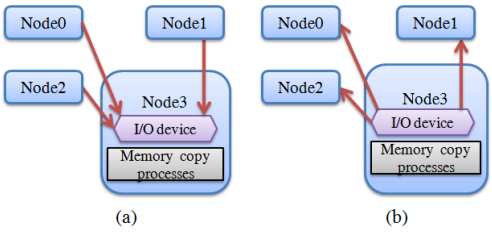


Fig. 9. Simulate I/O behavior with memory copy operation. Assume that there are 4 nodes in the system and I/O device is attached to node 3, so the target node for testing is node 3. (a) I/O device write simulation. Data sink locates at node 3, and data source varies among test cases (b) I/O device read simulation. Data source locates at node 3, and data sink varies among test cases.

in Figure 9(b). In this way, without involving I/O devices, we can emulate the I/O data transfer with only memory copy operations, and learn the I/O performance characteristics without costly I/O benchmark tests.

Figure 10 shows the I/O device write and read bandwidth performance with our proposed methodology. We can see that this result matches the I/O bandwidth performance levels we showed in Section IV-B. Some performance differences captured by our tool are not reflected in I/O test results. As stated in Section I-A, the I/O bandwidth performance can be impacted by various factors. In these cases, the I/O bandwidth bottleneck is not related the NUMA penalties. We categorize all the nodes into different classes in Table IV as the device write model and Table V as the device read model, according to their relative performance in Figure 10. The local and neighboring nodes are always be assigned to the first class, and the main task of our methodology is to classify the remote nodes. The performance models in both tables were first obtained by the proposed method, and then used to compare and analyze the actual I/O performance characteristics.

B. Implementation and application of the I/O performance model

The methodology used to model the performance of node 7 can also be generalized to other nodes in the host and other NUMA systems. We have implemented the methodology generally applicable as an *iomodel* test module, adding to

Algorithm 1: NUMA I/O performance modeling

Input: NODES TO CHARACTERIZE: k, MODEL TO OBTAIN: $model$

```
n ← numa_num_configured_nodes()
m ←num_configured_cores() / n
for i ← 1 to n do
    for p ← 1 to m do
        if mode = write then
            Allocate memsrc[p] in NUMA node i
            Allocate memsnk[p] in NUMA node k
        if mode = read then
            Allocate memsrc[p] in NUMA node k
            Allocate memsnk[p] in NUMA node i
    for p ← 1 to m do
        Create thread[p], bind to node k, copy from
        memsrc[p] to memsnk[p] for 100 times and
        record the average bandwidth
    for p ← 1 to m do
        thread_join(thread[p]);
if mode = write then
    Generate I/O device write performance model for
    node k
if mode = read then
    Generate I/O device read performance model for
    node k
return
```

TABLE IV
NUMA I/O BANDWIDTH PERFORMANCE MODEL FOR DEVICE WRITE(UNIT:GBPS)

Operation	Node ID	Class 1 6, 7	Class 2 0, 1, 4, 5	Class 3 2, 3
Proposed memcpy	Range Avg	46.5 – 55.9 51.2	42.9 – 46.9 44.5	26.0 – 27.3 26.6
TCP sender	Range Avg	19.6 – 20.9 20.3	20.0 – 21.0 20.4	16.2 – 16.3 16.2
RDMA_WRITE	Range Avg	23.3 – 23.3 23.3	23.2 – 23. 3 23.2	17.0 – 17.1 17.1
SSD write	Range Avg	28.6 – 29.1 28.8	28.1 – 28.9 28.5	17.9 – 18.0 18.0

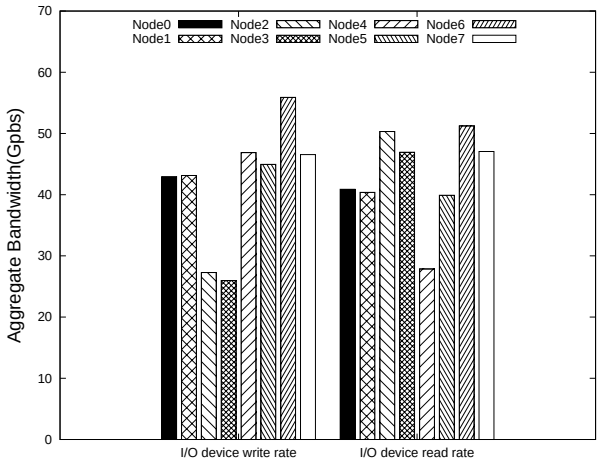


Fig. 10. Bandwidth performance model of node 7 by the proposed methodology

TABLE V
NUMA I/O BANDWIDTH PERFORMANCE MODEL FOR DEVICE READ(UNIT:GBIT/S)

Operation	Node ID	Class 1 6, 7	Class 2 2, 3	Class 3 0, 1, 5	Class 4 4
Proposed memcpy	Range Avg	47.1–51.2 49.1	46.9–50.3 48.6	39.9–40.9 40.4	27.9 27.9
TCP receiver	Range Avg	20.3–22.0 21.2	19.6–20.4 20.0	19.8–21.1 20.6	14.4 14.4
RDMA_READ	Range Avg	22.0–22.0 22.0	22.0–22.0 22.0	18–18.5 18.3	16.1 16.1
SSD read	Range Avg	34.7–34.7 34.7	32.3–32.9 33.1	29.7–30.9 30.1	18.5 18.5

in the standard *numademo* software package. The obtained performance models from our methodology and software, as illustrated in Table IV and Table V, can then bring in the following advantages:

- **Reduce the cost to capture the NUMA characteristics of the entire system.** In the performance model, we assume that all the nodes in the same class have similar bandwidth performance. While we want to understand the NUMA impact on system performance, instead of benchmarking all possible combinations, we can examine only one node from each class. For example, in the case of Table V, if we characterize the read speed of I/O devices attached to node 7, we only need to test four different NUMA setups from four classes. These representative tests will provide the same results as the whole testcase space (eight cases). Hence, the evaluation cost decreases by 50%.
- **Predict the overall performance in multi-user environment.** When an I/O device is shared by multiple users, it is highly possible that the data-access requests come from different NUMA nodes. Assume that we have achieved the performance model of the node attached to the I/O device using our method. Let BW_i be the average bandwidth performance of class i in the performance model, and $\alpha_i\%$ be the percentage of the data accesses that come from class i, where $i \in [1, \cdots, N]$, where N is the total number of classes. We predict the aggregate performance for the I/O device using the following model, $\hat{BW}_{io}$.

$$\hat{BW}_{io} = \sum_{i=1}^{N} \alpha_i\% \times BW_i \quad (1)$$

For example, in the case of RDMA_READ in Figure 6(b), if two processes transfer data from node 2 (of class 2) to the network card, and two other processes access from node 0 (of class 3), the overall bandwidth is estimated as $\hat{BW}_{io} = 50\% \times 18.036 + 50\% \times 21.998 = 20.017 Gbps$. We run this configuration with fio benchmark. Since the bandwidth performance is stable over the whole data transfer process, instead of showing the distribution of multiple repetitive test results, we demonstrate the average aggregate bandwidth while transferring a large amount of data ($400 GBytes$ per process), which is $19.415 Gbps$. Hence, the relative error is $\varepsilon = \frac{|20.017-19.415|}{19.415} \times 100\% = 3.1\%$. This example shows that our model can estimate the overall bandwidth of the I/O devices in a multi-user scenario.
- **Assist resource schedulers on NUMA systems.** With application-layer NUMA scheduling, a programmer should have enough information about a system's NUMA characteristics. With this, numerous scheduling algorithms [28], [29], [30] can be applied to modern high-end NUMA systems. For example, as we mentioned before, in a multi-user environment, binding all I/O tasks to their local node will lead to severe performance degradation due to the contention of shared resource. With the knowledge of our performance model, the task scheduler can distribute application processes to nodes in the same class or the classes with the same performance. For example, in the case of RDMA_WRITE in Figure 6(a), after our characterization, we know that class 1 and class 2 have almost identical performance. Therefore, instead of allocating all application processes to node 7 only, we can evenly split the task processes among all nodes in class 1 and class 2. Therefore, the overall performance will be improved due to much less contention for shared resources.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the problem of accurately characterizing and predicting I/O bandwidth performance in modern high-end NUMA systems. Directly applying a software benchmark to characterize memory and I/O hardware might lead to unexpected performance inconsistency among multiple tests, and can potentially generate a daunting experimental load. We illustrated the reasons why existing NUMA-related metrics and tools cannot solve the problem by quantitative comparisons. We then proposed our own methodology and software to obtain the I/O bandwidth performance model without involving the actual I/O hardware to be modeled

and costly I/O benchmarking process. To the best of our knowledge, our work is the first attempt to propose an I/O performance model based simple memory operations for NUMA systems. The experimental results confirmed that our empirical method can predict the I/O bandwidth characteristics among various NUMA architectures. At last, we provided concrete examples to illustrate the applications of the performance models produced by our methodology.

There are two directions of our future work: 1) we will continue working on the mechanisms of placing and migrating parallel I/O threads for data-intensive applications based on the result of our characterization methodology, and 2) we will study more delicate issues such as architectural details leading to performance asymmetry and tradeoffs between data locality and resource contention.

ACKNOWLEDGMENT

The authors are grateful to the generous hardware donation from Mellanox Technologies Inc and LSI Corporation. The authors have benefited from numerous technical discussions with David McMillen from System Fabric Works Inc and Todd Wilde from Mellanox Technologies Inc. This research is supported by United States Department of Energy, Grant No. DE-SC0003361. The contract is funded through the American Recovery and Reinvestment Act of 2009.

REFERENCES

[1] A. Kayi, E. Kornkven, T. El-Ghazawi, S. Al-Bahra, and G. Newby, "Performance evaluation of clusters with ccNUMA nodes - a case study," in *The 10th IEEE International Conference on High Performance Computing and Communications*, September 2008, pp. 320–327.

[2] J. Shakshober and L. Woodman, "Red Hat Enterprise Linux performance and scalability," 2012. [Online]. Available: http://rhsummit.files.wordpress.com/2012/03/shak_larry_perf_analysis_and_tuning.pdf

[3] O. Dumitru, R. Koning, and C. D. Laat, "40 Gigabit Ethernet: prototyping transparent end-to-end connectivity," in *The TERENA Networking Conference 2011 (TNC 2011)*, 2011.

[4] K. Spafford, J. Meredith, and J. Vetter, "Quantifying NUMA and contention effects in multi-GPU systems," in *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, 2011, pp. 1–7.

[5] S. Moreaud and B. Goglin, "Impact of NUMA effects on high-speed networking with multi-Opteron machines," in *The 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2007, pp. 24–29.

[6] Mellanox Technologies Inc, "Connect-IB: Architecture for scalable high performance computing." [Online]. Available: http://www.mellanox.com/related-docs/applications/SB_Connect-IB.pdf

[7] LSI Corporation, "Nytro WarpDrive WLP4-200 application acceleration card." [Online]. Available: http://www.lsi.com/products/storagecomponents/Pages/NytroWarpDriveWLP4-200.aspx

[8] Z. Majo and T. Gross, "Memory system performance in a NUMA multicore multiprocessor," in *Proceedings of the 4th Annual International Conference on Systems and Storage*, 2011, pp. 1–10.

[9] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for NUMA-aware contention management on multicore systems," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, 2011.

[10] B. M. Tudor, Y. M. Teo, and S. See, "Understanding off-chip memory contention of parallel programs in multicore systems," in *2011 International Conference on Parallel Processing (ICPP 2011)*, September 2011, pp. 602–611.

[11] L. L. Pilla, C. P. Ribeiro, D. Cordeiro, C. Mei, A. Bhatele, P. O. A. Navaux, F. Broquedis, and L. V. K. Jean-Franccois Mehaut, "A hierarchical approach for load balancing on parallel multi-core systems," in *2012 41st International Conference on Parallel Processing (ICPP 2012)*, September 2012, pp. 118–127.

[12] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007.

[13] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the AMD Opteron processor," *IEEE Micro*, pp. 16–29, 2010.

[14] S. Akram, M. Marazkis, and A. Bilas, "NUMA implications for storage I/O throughput in modern servers," in *3rd Workshop on Computer Architecture and Operating System co-design (CAOS'12)*, 2012.

[15] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, December 1995.

[16] A. Kleen, "A NUMA API for Linux," 2004. [Online]. Available: http://halobates.de/numaapi3.pdf

[17] S. Blagodurov and A. Fedorova, "User-level scheduling on NUMA multicore systems under Linux," in *Linux Symposium*, 2011.

[18] P. McCormick, R. K. Braithwaite, and W. Feng, "Empirical memory-access cost models in multicore NUMA architectures," Los Alamos National Laboratory (LANL), Tech. Rep., January 2011.

[19] The Open MPI Development Team, "The Portable Hardware Locality (hwloc) software package," 2012. [Online]. Available: http://www.open-mpi.org/projects/hwloc/

[20] Advanced Micro Devices Inc, "HyperTransport 3.0 Specification," 2012. [Online]. Available: http://www.hypertransport.org/default.cfm?page=HyperTransportSpecifications3

[21] "InfiniBand architecture specification release 1.2.1 Annex A16: RoCE," InfiniBand Trade Association, Tech. Rep., April 2010.

[22] Mellanox Technologies Inc, *Performance Tuning Guidelines for Mellanox Network Adapters*.

[23] Advanced Micro Devices Inc, "AMD Opteron 6200 series processors linux tuning guide," 2012. [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/51803A_OpteronLinuxTuningGuide_SCREEN.pdf

[24] J. Axboe, "Flexible I/O Tester." [Online]. Available: http://freecode.com/projects/fio

[25] Y. Ren, T. Li, D. Yu, S. Jin, T. Robertazzi, B. Tierney, and E. Pouyoul, "Protocols for wide-area data-intensive applications: Design and performance issues," in *Supercomputing 2012*, November 2012.

[26] Advanced Micro Devices Inc, *BIOS and Kernel Developer's Guide (BKDG) For AMD Family 11h Processors*, 2008. [Online]. Available: http://support.amd.com/us/Processor_TechDocs/41256.pdf

[27] R. K. Braithwaite, W. Feng, and P. McCormick, "Automatic NUMA characterization using Cbench," in *Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering*, 2012.

[28] W. J. Bolosky, M. L. Scott, R. P. Fitzgerald, R. J. Fowler, and A. L. Cox, "NUMA policies and their relation to memory architecture," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

[29] H. Li, S. Tandri, M. Stumm, and K. C. Sevcik, "Locality and loop scheduling on NUMA multiprocessors," in *International Conference on Parallel Processing (ICPP 1993)*, Augest 1993, pp. 140–147.

[30] Z. Majo and T. Gross, "Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead," in *Proceedings of the International Symposium on Memory Management*, 2011, pp. 11–20.