Modeling Communication Costs in Blade Servers

Qiuyun Wang, Benjamin C. Lee
Duke University
Department of Electrical and Computer Engineering
{qiuyun.wang, benjamin.c.lee}@duke.edu

ABSTRACT
Datacenters demand big memory servers for big data. For blade servers, which disaggregate memory across multiple blades, we derive technology and architectural models to estimate communication delay and energy. These models permit new case studies in refusal scheduling to mitigate NUMA and improve the energy efficiency of data movement. Preliminary results show that our model helps researchers coordinate NUMA mitigation and queueing dynamics. We find that judiciously permitting NUMA reduces queueing time, benefiting throughput, latency and energy efficiency for datacenter workloads like Spark. These findings highlight blade servers’ strengths and opportunities when building distributed shared memory machines for data analytics.

Categories and Subject Descriptors
C.0 [Computer Systems Organization]: General—System architectures; C.4 [Computer Systems Organization]: Performance of Systems—Design studies, Modeling techniques; B.3 [Hardware]: Memory Structures—Shared memory

Keywords
NUMA, technology models, energy and delay, blade servers, scheduling, communication cost

1. INTRODUCTION
Blade servers provision abundant memory in a dense form factor and their distributed shared memory are well suited to big data applications. Researchers have prototyped or emulated blade architectures [11, 6] to understand their potential and to demonstrate key capabilities, such as fine-grained access and address translation. Beyond specific designs, however, researchers must assess sensitivity to technology parameters and explore server organizations. Unfortunately, existing experimental methods lack the required flexibility.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HotPower’15 October 04-07, 2015, Monterey, CA, USA
©2015 ACM. ISBN 978-1-4503-3946-9/15/10 ...$15.00
DOI: http://dx.doi.org/10.1145/2818613.2818743.
Memory Communication. The memory controller is integrated into the processor die and responds to last-level cache misses. Memory latency is affected by array latency, queuing delays, and request scheduling. The controller can schedule memory requests for a mix of latency, throughput, and fairness targets. Sophisticated scheduling mitigates row buffer misses and bank conflicts, which add to memory latency.

DRAMs consume dynamic energy due to precharges, activations, reads, and writes. In addition, DDRx interfaces, which include delay-locked loops and on-die termination, draw static current regardless of channel utilization. Thus, high-performance memories are energy-disproportional for tasks with modest bandwidth demands [13]. 2Gb DDR3 with SDQs strikes a balance between chip capacity and I/O width [13] and dissipate approximately 2W per GB. A typical server has 8GB per channel, 2 channels per processor, and 4 processors per blade. In total, 64GB of DRAM dissipates 128W.

Inter-Processor Communication. Multiple processors are integrated into a blade. They share a physical address space and can support coherent access to shared memory. Both the local memory controller and the interconnect controller observe a last-level cache miss. The latter uses the memory address to identify and route a memory request destined for another processor’s controller.

A remote memory request may require one or two hops, which introduces latency and energy costs in addition to those from DRAM. The interconnect uses serial, point-to-point links for high data rate and low latency. We use HyperTransport (HT) for the inter-processor connection [5]. HT transmitter logic requires 18ns to encode contents and add headers. Packet transmission and link interface circuitry adds 14ns. Receiver logic requires another 18ns. In total, the packetized request requires 50ns to reach a remote memory controller. With round-trip overheads and DRAM access delay, remote data access requires at least 150ns.

To assess energy costs, we examine serial links and their interfaces. A serial link requires serializer/deserializer (SerDes) circuitry at its endpoints to convert data between parallel and serial interfaces. This circuitry determines the energy cost per bit transferred. How this cost translates into system power depends on the number of processors, the number of links between them, and the data rates of those links [4].

HT links connect two processors with 16 lanes, each implemented with paired serial links for differential signaling. Bi-directional communication requires two paired links since a serial link is unidirectional. Thus, 16 lanes require 64 links and 128 SerDes. Each SerDes consumes 10pJ per bit transferred and can transfer up to 6.4Gb per second [7].

We estimate power by multiplying the number of interfaces, the transfer rate and the cost per transfer – power is 8.2W per path and a server with five paths dissipates up to 40W.

Inter-Blade Communication. Blades share an address space and communicate via a backplane interconnect. Bridges and switches perform address translation and route memory requests to the appropriate blade [6, 18]. We describe a blade architecture in which a processor can access a remote blade’s memory with load/store instructions via PCIe. The memory controller, PCIe root complex, and non-transparent bridges (NTBs) are all integrated into the processor die [6]. Each inter-blade connection begins and ends with an NTB bridge or switch. A typical 64b PCIe transmission requires 240ns of which 50ns is attributed to DRAM [5]; the remaining 190ns is the round trip PCIe link transmission delay.

Inter-blade communication also depends on the number of HT hops. For example, six PCIe links connect twelve NTBs in Figure 1. A memory request may traverse HT to reach the correct bridge and traverse PCIe to reach the correct blade. We calculate the expected HT delays on sending and receiving blades, incurred in addition to PCIe and DRAM latencies. With round-trip overheads, accessing data on a remote blade may require 410ns, including 50ns for DRAM, 190ns for PCIe, and 170ns for expected HT delay.

Energy costs increase with blade connectivity. In our example, each uni-directional connection is 16 lanes-wide and a bi-directional connection requires 32 lanes that dissipate 4.8W. Each lane dissipates 150mW and transfers data at 0.5GB/s [8]; energy cost is 37.5pJ/bit. Bridge logic and SerDes circuitry dissipate an additional 2.5W. In total, an inter-blade link dissipates 10W for 32 lanes and two bridges.

We consider connectivity between 4 blades, each with 4 processors and integrated bridges. These NTBs support back-to-back PCIe connections between pairs of blades. A fully-connected network requires 6 connections and 12 NTBs, which in total dissipate up to 60W.

In summary, our technology models are accurate and consistent with prior measurements. Memory estimates use DRAM specifications [15]. Inter-Processor estimates use HT technology parameters [5] to calculate latency from packet processing and transmission. Estimates align with NUMA measurements [2]. Inter-processor power numbers (40W at peak and 10-20W at typical utilization) are consistent with industry measurements [1]. Inter-Blade estimates are consistent with emulation parameters for disaggregated memory [10]. Power estimates are derived from PCIe bridge and link specifications [16, 7].

3. CASE STUDIES IN MANAGING NUMA

We illustrate opportunities for parameterized design and management with two case studies that deploy our technology models. First, we adapt queuing policies to the costs of varied communication paths. Second, we compare the energy efficiency of remote execution against migration.

Execution Model. Run-time systems rely on task queues to produce parallelism while preserving a programming model’s clean abstractions (e.g., MapReduce, Spark, GraphLab). The task at the head of the queue is likely to find its data already in main memory because of caching or pre-fetching.

<table>
<thead>
<tr>
<th>Technology</th>
<th>DDR3</th>
<th>HT3.1</th>
<th>PCIe 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 2Gb, x8</td>
<td>PTP</td>
<td>PTP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Rate (GB/s per lane)</th>
<th>0.2</th>
<th>0.8</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lanes</td>
<td>x64</td>
<td>x32</td>
<td>x16</td>
</tr>
</tbody>
</table>

| Uni-directional B/W (GB/s) | 12.8 | 25.6 | 8.0 |

| Transfer Energy (pJ/bit)   | 160 (0.20% util.) | 36.0 | 37.5 |
|                           | 70.0 (0.10% util.) |

| Latency (ns)              | 50-100 | 100 RT | 190 HT |

Table 1: Summary of technology models and estimates
Dequeue Task
first, check retrial queue
then, check task queue
Local ?
does available core's 
memory contain task data?
next task
yes no
yes
no
  task 
queue
retrial 
queue
retrying 
tasks
new
tasks

Figure 2: Refusal policies and queue management.

For example, Spark caches data from the current iteration of a machine learning kernel to ensure its availability for the next one.

However, non-deterministic queueing complicates the coordination of task scheduling and data placement; the data placement mechanism cannot predict which core will become available at a certain time. A lack of coordination exposes NUMA in blade servers. Tasks must navigate multiple levels of NUMA and still guarantee service quality. We look beyond the bimodal locality classification (e.g., local versus remote) and examine distance to data when scheduling tasks with NUMA.

**Refusal Scheduling.** We draw inspiration from delay scheduling [20], which improves storage locality for MapReduce tasks. Our case studies present policies that dictate whether a queued task should refuse execution on an available core due to NUMA.

Figure 2 illustrates refusal scheduling. Arriving tasks enter a queue. When a core becomes available, the scheduler determines this core’s proximity to data required by the next task (e.g., address of task’s vertex given graph in distributed memory). The scheduler’s refusal policy selectively refuses and permits NUMA. Refusing tasks enter a high-priority retrial queue. Queues prioritize tasks with earlier arrival times for FIFO fairness. A refusal limit is used to avoid starvation. We consider four refusal policies for multiple NUMA levels (e.g., Figure 1).

- **Local Execution (Local).** Task runs on processor for which its data is local. Otherwise, task refuses.
- **Inter-Processor 1-Hop Execution (IP-1).** Task favors local execution. It also accepts 1-hop inter-processor communication.
- **Inter-processor 2-Hop Execution (IP-2).** Task prefers running on blade for which its data is local. It refuses inter-blade communication.
- **Inter-blade Execution (IB).** Task executes on any available core. It favors cores closer to its data.

A refusal policy balances queueing and service time. The optimal policy depends on distance to data, sensitivity to NUMA, server utilization, as well as performance and efficiency goals.

### 4. EXPERIMENTAL METHODOLOGY

We link multiple simulators to coordinate the study of NUMA and queueing dynamics. Processor/memory simulation quantifies task sensitivity to NUMA. Queueing simulation quantifies system latency and throughput.

**Processor Simulation.** We use Marssx86 and DRAMSim2 for processor and memory respectively. We simulate a 4-way OoO core (2GHz) with 128KB iCache and dCache, 2M L2 cache, and DDR3 with a 667MHz bus. Each benchmark is simulated on the region of interest for 200 million instructions; the results are shown in Table 2. When accessing remote memory, we add inter-processor and inter-blade latencies according to the technology models in §2. We focus on communication for heap data, assume that local memory holds code and stack. The simulator identifies address regions for code, stack, and heap to distinguish remote requests from local ones.

**Queueing Simulation.** We implement a discrete event simulator, modeled after BigHouse [14], to evaluate scheduling policies and queueing dynamics within a blade server (e.g., Figure 1). The queueing simulator uses service times from processor simulations, which account for NUMA, to estimate response times in G/G/k queues. It tracks each task and reports summary statistics for service quality (e.g., 95th percentile response time). We implement the Local, IP-1, IP-2 and IB policies and simulate them for hundreds of millions of tasks. We assume the data distribution and demand is uniformly and randomly distributed among a blade server’s processors.

**Benchmarks.** Table 2 summarizes tasks’ demand for remote memory bandwidth (Extremely Memory-Intensive, Memory Intensive, Compute Intensive) and latency penalties from NUMA.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Bandwidth</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MI/CI,MB/s</td>
<td>HT-1 PCIe</td>
</tr>
<tr>
<td>S1</td>
<td>Word count MI</td>
<td>264</td>
<td>1.17x</td>
</tr>
<tr>
<td>S2</td>
<td>Logistic reg. MI</td>
<td>242</td>
<td>1.17x</td>
</tr>
<tr>
<td>S3</td>
<td>Pagerank MI</td>
<td>276</td>
<td>1.22x</td>
</tr>
<tr>
<td>S4</td>
<td>Transitive closure MI</td>
<td>235</td>
<td>1.19x</td>
</tr>
<tr>
<td>S5</td>
<td>Alternating least squares MI</td>
<td>231</td>
<td>1.16x</td>
</tr>
<tr>
<td>S6</td>
<td>K-means MI</td>
<td>283</td>
<td>1.22x</td>
</tr>
<tr>
<td>S7</td>
<td>Pi CI</td>
<td>21</td>
<td>1.01x</td>
</tr>
</tbody>
</table>

**Phoenix MapReduce**

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Bandwidth</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MI/CI,MB/s</td>
<td>HT-1 PCIe</td>
</tr>
<tr>
<td>M8</td>
<td>Word count MI</td>
<td>111</td>
<td>1.10x</td>
</tr>
<tr>
<td>M9</td>
<td>Histogram CI</td>
<td>42</td>
<td>1.02x</td>
</tr>
<tr>
<td>M10</td>
<td>String match CI</td>
<td>28</td>
<td>1.04x</td>
</tr>
<tr>
<td>M11</td>
<td>Linear regression CI</td>
<td>7</td>
<td>1.0x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parsec</th>
<th>Name</th>
<th>Bandwidth</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>P12</td>
<td>Facesim EMI</td>
<td>2159</td>
<td>1.39x</td>
</tr>
<tr>
<td>P13</td>
<td>Dedup CI</td>
<td>50</td>
<td>1.09x</td>
</tr>
<tr>
<td>P14</td>
<td>Bodytrack CI</td>
<td>35</td>
<td>1.02x</td>
</tr>
<tr>
<td>P15</td>
<td>Vips CI</td>
<td>23</td>
<td>1.01x</td>
</tr>
<tr>
<td>P16</td>
<td>Ferret CI</td>
<td>73</td>
<td>1.04x</td>
</tr>
<tr>
<td>P17</td>
<td>Raytrace CI</td>
<td>19</td>
<td>1.02x</td>
</tr>
<tr>
<td>P18</td>
<td>Swaptions CI</td>
<td>5</td>
<td>1.00x</td>
</tr>
<tr>
<td>P19</td>
<td>Streamclusters CI</td>
<td>4</td>
<td>1.08x</td>
</tr>
<tr>
<td>P20</td>
<td>Blacksholes CI</td>
<td>3</td>
<td>1.08x</td>
</tr>
</tbody>
</table>

Table 2: Benchmarks, their demand for remote memory bandwidth (Extremely Memory-Intensive, Memory Intensive, Compute Intensive) and latency penalties from NUMA.
Figure 3: (a) Throughput for varied benchmarks indexed in Table 2. (b, c) 95th percentile response times normalized to IB policy, which permits all NUMA, under high and low server load.

5. EVALUATION

We use our models and simulators to evaluate case studies in NUMA mitigation. We find that the optimal refusal policy varies according to a task’s memory intensity and system utilization. In addition, we discover that permitting fine-grained NUMA is more energy-efficient than coarse-grained page migration.

**Throughput.** We analyze throughput by (1) simulating a task and quantifying its performance under three communication scenarios—local, inter-processor, and inter-blade; (2) quantifying the maximum sustainable throughput for the task stream on a blade node. We find that avoiding NUMA reduces task service time and increases throughput.

Figure 3(a) shows how throughput improves as the refusal policy permits more NUMA. A policy that favors local execution will lower service time and increase service. This effect is most pronounced for memory-intensive workloads (S1-S6, M8, P12), which suffer more from NUMA penalties. Ranking policies by increasing throughput gives: IB, IP-2, IP-1, Local.

**Response Time.** Judiciously permitting NUMA reduces queuing delay and response time. Permissive policies, which allow execution when a core becomes available regardless of distance to data, mean less time in arrival and retrial queues. However, latency is not determined by NUMA policy alone. Policy interacts with server load and a task’s NUMA sensitivity. We simulate queuing dynamics for highly and lightly loaded servers; system load increases with task arrival rate.

Figure 3 reports the 95th percentile for response time. NUMA execution can be beneficial, even when tasks are memory-intensive (S1-S6, M8, P10) and server load is high. For example, IP-1 and IP-2 consistently perform better than Local because they improve tasks’ likelihood of de-queueing. De-queueing quickly is even more beneficial for compute-intensive tasks (S7, M9, M11, P13-P20), which require little data movement. Refusing NUMA execution does not reduce service time and only increases queuing time. Therefore, compute-intensive tasks can adopt IB, which permits load/stores to another blade’s memory, with little penalty.

When the server is lightly loaded, many cores are available, queuing time is less important, and service time dominates response time (Figure 3(c)). Refusing NUMA execution only marginally improves response time. Local outperforms IB and IP-2 by less than 20% for memory-intensive tasks and by 2%-5% for compute-intensive ones.

**Comparison to Data Migration.** Task scheduling policies that permit NUMA exploit fine-grained load/store access to remote memory. Remote access retrieves the data needed to fill a cache line instead of migrating entire pages from remote locations into local ones. Although migration is fast, its energy costs are high. We compare the energy costs of remote access and data migration. Our results show that migration energy is 1.5-5.0× greater than that of remote access for memory-intensive workloads.

Energy cost depends on the amount of data transferred and the links used. The processor and queueing simulations report data transferred by remote accesses and tasks that use each link. To estimate the amount of data migrated, we multiply the page size (4KB) by the number of unique pages accessed by a program. Let $D_{\text{page}}$ denote the amount of data migrated and $D_{\text{access}}$ denote the amount of data supplied to cache lines.

$$E_M = D_{\text{page}} (2E_{\text{mem@100}} + E_{\text{link}}) + D_{\text{access}} E_{\text{mem@20}}$$  (1)

Equation 1 estimates page migration energy. Migration accesses DRAM twice, to read from remote memory and to write into local memory. These memory transfers may traverse inter-processor and inter-blade links. The cost of transferring data through DRAM at full bandwidth $E_{\text{mem@100}}$ is 70pJ/bit. The link cost $E_{\text{link}}$, based on the expected number of inter-processor hops, is 110pJ/bit—see Table 1. Filling cache lines is expensive under low channel utilization and $E_{\text{mem@20}} = 110pJ/bit$ [13].

$$E_{\text{RT}} = D_{\text{access}} (E_{\text{link}} + E_{\text{mem@20}})$$  (2)

Equation (2) estimates the cost of remote memory accesses. We multiply cache line size (64B) by the number of remote last-level cache misses to estimate the amount of data accessed, $D_{\text{access}}$.

**Energy Efficiency.** Figure 4 shows that page migration energy is 1.5-5.0× that of remote access, which transfers fine-grained cache lines. However, if migrated data is re-used, we might expect migration to outperform remote execution. The break-even points, where Equations 1 and 2 are equal, are expressed in the number of re-uses.

- Inter-Blade $D_{\text{access}}/D_{\text{page}} = 2.2\times$
- Inter-Processor two-hop $D_{\text{access}}/D_{\text{page}} = 2.9\times$
8. REFERENCES