A Novel System Architecture for Web Scale Applications Using Lightweight CPUs and Virtualized I/O *

Kshitij Sudan[∞†] Saisanthosh Balakrishnan[§‡] Sean Lie[§] Min Xu[§]
Dhiraj Mallick[§] Gary Lauterbach[§] Rajeev Balasubramonian[∞]

[∞]University of Utah
{kshitij,rajeev}@cs.utah.edu

[§]SeaMicro Inc. (now AMD Inc.)
{sai,sean,min,dhiraj,gary}@seamicro.com

Abstract

Large web-scale applications typically use a distributed platform, like clusters of commodity servers, to achieve scalable and low-cost processing. The Map-Reduce framework and its open-source implementation, Hadoop, is commonly used to program these applications. Since these applications scale well with an increased number of servers, the cluster size is an important parameter. Cluster size however is constrained by power consumption. In this paper we present a system that uses low-power CPUs to increase the cluster size in a fixed power budget. Using low-power CPUs leads to the situation where the majority of a server's power is now consumed by the I/O sub-system. To overcome this, we develop a virtualized I/O sub-system where multiple servers share I/O resources. An ASIC based high-bandwidth interconnect fabric, and FPGA based I/O cards implement this virtualized I/O. The resulting system is the first production quality implementation of ***cluster-in-a-box*** *that uses low-power CPUs. The unique design demonstrates a way to build systems using low-power CPUs, allowing a much larger number of servers in a cluster in the same power envelope. To overcome software inefficiency and increase the utilization of virtualized disk bandwidth, optimizations necessary for the operating system are also discussed. We built hardware based on these ideas and experiments on this system show a 3X average improvement in performance-per-Watt-hour compared to a commodity cluster with the same power budget.*

1 Introduction

With a growing number of services being delivered over the Internet, the amount of data being generated and analyzed has grown tremendously. The infrastructure required to process these large datasets has also grown accordingly. To keep the costs of processing data low, new hardware and software frameworks have been developed. The most notable among these is the MapReduce (MR) framework advanced by Google [13].

The goal of the MR framework is to leverage the cost advantages of commodity servers while providing reliable execution of applications that process large amounts of data. It also facilitates application development by abstracting away low-level resource management details of large distributed systems. MR has become popular with web companies that need a "scale-out" architecture. Scale-out allows companies to grow their infrastructure by simply adding more commodity servers to an existing cluster. The Apache Hadoop project [7] is an open source implementation of Google's MR framework and is being used by companies such as Facebook [11], Yahoo! [2], and Amazon's Elastic MapReduce service [1]. For this paper we will discuss the MR framework with Hadoop as our implementation.

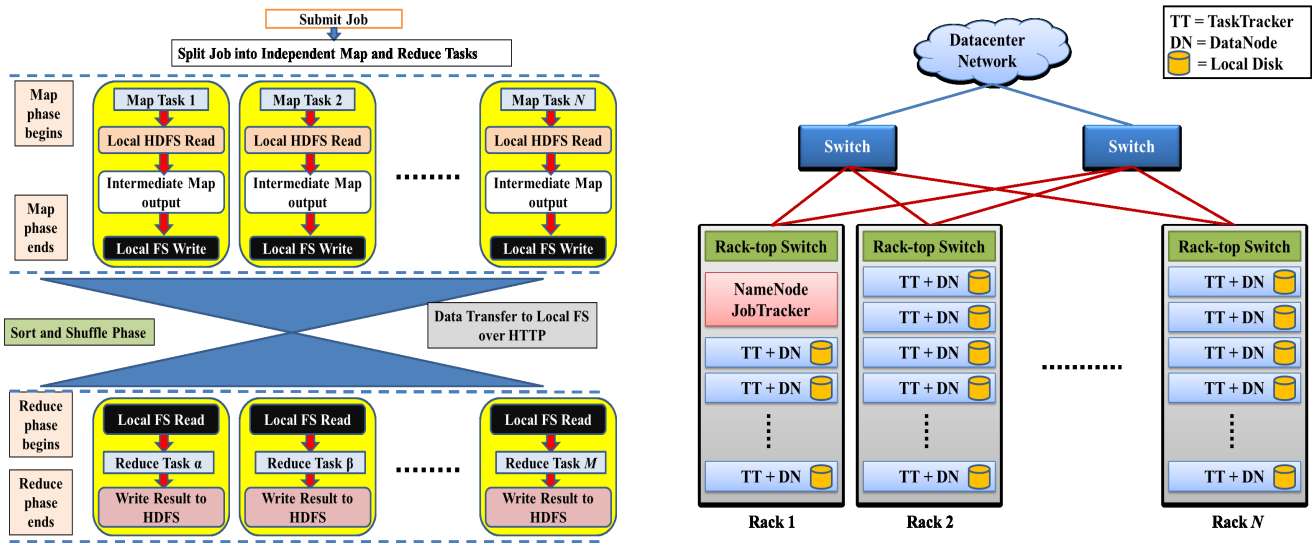
The MR framework splits up a job into small tasks that are executed in parallel on a cluster built from commodity servers. MR is supported by a distributed filesystem – Google File system (GFS) [16], or the Hadoop Distributed File System (HDFS) [3]. The distributed filesystem stores data reliably across the cluster, where failures can be frequent. It does so by slicing the data into small chunks (typically 64 MB) and replicating the slices across multiple cluster nodes. The data slices are stored on the local disks of each node. The framework schedules tasks that process a given slice of data to nodes in the cluster that store that slice locally. Thus the distributed filesystem and the MR engine work together to move compute closer to data.

One of the advantages of using the MR framework is that it allows scaling of applications by simply adding more nodes to the cluster [21]. The cluster size however is constrained by the power delivery setup at datacenters housing these clusters [15]. As a result, application performance

*This work was supported by SeaMicro Inc. (now AMD Inc.), NSF grant CCF-0916436, and the University of Utah.

†Author is currently at Samsung Austin R&D Center. This work was performed while the author was an intern at SeaMicro Inc.

‡Author is currently at Violin Memory Inc.



(a) Map-Reduce Computation Model

(b) Hadoop Cluster Organization

Figure 1. MapReduce Computational Model and Hadoop Cluster Organization

per unit of power is an important metric. At the same time, for large clusters, the cost of operating the cluster is significant and a large fraction of the operating cost is the energy cost. The cost of electricity to power a large cluster over its lifetime already exceeds its capital costs in some cases [28]. In this context, the total cost of ownership (TCO) of the cluster becomes important. TCO equals the sum of capital and operational expenses over the cluster's lifetime. As a result of these factors we focus on performance/TCO, and performance/Watt metrics to compare cluster designs. If the power delivery constraints are encountered first, the perf./Watt metric measures the maximum performance achievable with a given power delivery setup. If power delivery is not a show-stopper, the perf./TCO metric measures the cost incurred for a defined performance target.

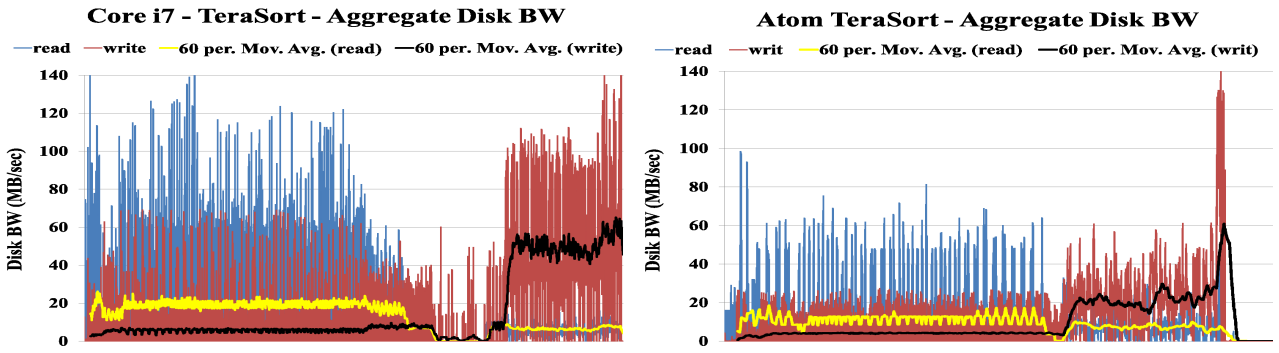
Using low-power, energy efficient CPUs is a promising way to build clusters that scale to a large number of servers in a limited power budget [9, 26]. Designing servers with such CPUs however is not simple. When using these CPUs, server power consumption is dominated by the I/O subsystem. At the same time, due to overheads in the software stack, the MR applications cannot fully utilize the provisioned I/O resources like the disk bandwidth. Furthermore, the power consumption of fixed system components like the power supply, fans, etc., also becomes dominant.

Based on these observations we present a new system architecture that better matches the characteristics of MR workloads and allows scaling to larger cluster sizes in a fixed power budget. This is achieved by using low-power CPUs and virtualizing the I/O sub-system – specifically the disk and the Ethernet. Our design also amortizes the power overheads of fixed system components like fans and power supplies. We show that virtualizing the storage sub-system balances the compute-to-disk bandwidth ratio, leading to improvements in performance/TCO and performance/Watt. Our primary contributions in this work are:

- **Balanced Resource Provisioning via Virtualized I/O:** Using low-power CPUs allows building larger clusters within a fixed power budget. A low-power CPU like the Intel Atom and its chipset consumes ∼11 Watts, while disks and Ethernet cards consume between 5-25 Watts each. The combination of low-power CPUs and software overheads however leads to sub-optimal use of the I/O resources. This leads to over-provisioned I/O resources that consume a large fraction of the server power. Sharing the I/O resources among multiple servers via virtualized I/O balances both the provisioned resources and server-level power distribution. We develop a hardware mechanism to virtualize the I/O sub-system such that it matches its power consumption with utilization.
- **Tackling Fixed Power Consumption:** Our architecture demonstrates how a system with low-power CPUs can amortize fixed power consumption by distributing the power consumption of commodity fans and power supplies over multiple servers. This is a major concern for servers with low-power CPUs and we show how to overcome this challenge.

2 Map-Reduce Background

The MapReduce (MR) framework [13] was designed with the goals of allowing application scaling, and ability to process large datasets. An MR job is automatically split into smaller tasks which are then scheduled in parallel on a cluster of servers. An MR job executes in three phases:



(a) Executing on a Core i7 based cluster

(b) Executing on an Atom based cluster

Figure 2. Disk bandwidth utilization by Core i7 and Atom based clusters while executing TeraSort benchmark. The overlaid lines are 1 minute averages of individual data points.

Map, *Shuffle*, and *Reduce*. Figure 1(a) shows the data flow between the different phases of a job. Each map task is assigned a partition of the input data [29]. The map task is then run on a cluster node that is in close proximity to its input data on the distributed file system (HDFS). The map task writes its intermediate output to the local file system of the cluster node. The shuffle phase then executes between the map and reduce phases. It reads the map outputs and sends them to the reducer nodes via HTTP. The reduce task is then run on this data to generate the final output which is written to the HDFS. As can be observed, the performance of a MR job is not only tied to the CPU performance but also to the I/O performance.

At the system level, Figure 1(b) shows the organization of a typical Hadoop cluster. There are four daemons that execute on cluster nodes - *NameNode*, *DataNode*, *JobTracker*, and *TaskTracker*. The *NameNode* maintains the HDFS filesystem. It only stores filesystem metadata and acts as a gatekeeper for reads and writes to the HDFS. When reading data from HDFS, the requester contacts the *NameNode* which provides a list of *DataNodes* containing the data. *DataNodes* then deliver the data over HTTP directly to the requester. The *JobTracker* controls job submission and scheduling. It splits a job into smaller tasks and schedules the tasks on cluster nodes running the *TaskTracker* daemons. The tasks are scheduled on a best-effort basis such that each task retrieves its input data from the *DataNode* daemon running on the same cluster node. This amounts to moving compute closer to data – an important property when working on large datasets.

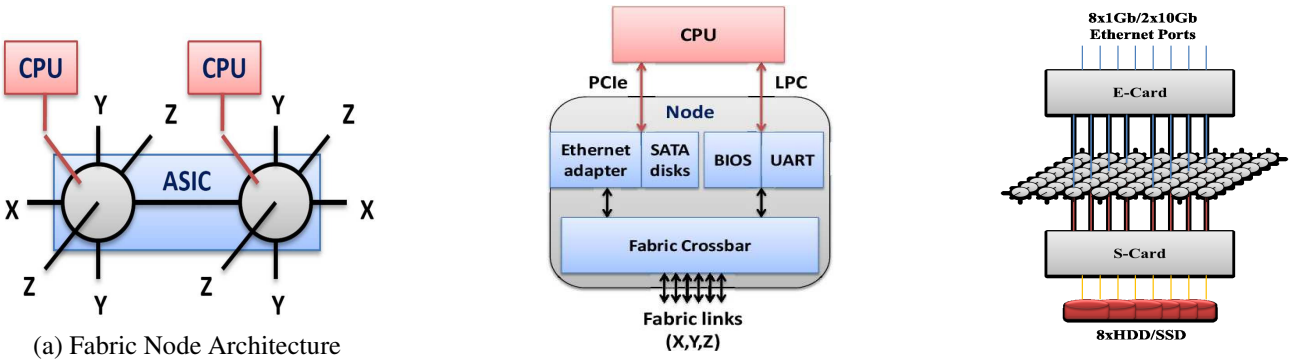
3 Motivation

Our system design is motivated by two considerations: (a) *increasing the cluster size within a fixed power budget*, and (b) *balanced resource provisioning to improve system efficiency*.

Increasing Cluster Size in Fixed Power Budget: One of the defining features of web-scale applications is their ability to scale with an increase in hardware resources [21]. A larger cluster size typically means smaller application execution time. However, the cluster size is limited by the power delivery systems [15, 17]. To stay within power limits and build extremely large clusters, one option is to use low-power CPUs. Nearly fifteen low-power CPUs like the Intel Atom N570 [4] (consuming 8.5 W each) can be substituted for one high performance CPU like the Xeon E7-8870 [5] (rated power of 130 W) in the same power budget. Azizi et al. [10] also showed that lightweight architectures are also more energy efficient since they operate at the most efficient point on the energy-performance curve among various CPU architectures. Low-power CPUs therefore deliver superior performance/Watt.

Balanced Resource Provisioning: Web-scale applications process large amounts of data and are typically I/O intensive. Disk and network bandwidth are therefore critical design aspects for a Hadoop cluster. Overheads in the Hadoop framework, JVM, and the OS however limit the usable disk bandwidth. Figure 2 shows the under utilization of disk bandwidth in Core i7 and Atom based clusters. Details of the workload and platform are provided in Section 5. Bandwidth usage is plotted on the Y-axis and execution time on the X-axis. Bandwidth was measured at 1 sec intervals and the 60 sec moving average of these measurements is overlaid in the figure. The average bandwidth usage for both Core i7 and Atom cluster nodes peaks at 60 MB/sec, and the sustained average varies between 20-50 MB/sec.

These values are significantly lower than the raw bandwidth each CPU can drive from the disk. Core i7 based systems can drive a sustained disk read bandwidth of 112 MB/sec, while Atom servers can drive upto 80 MB/sec of read bandwidth. Note that these bandwidth limits are also a function of the disk controller, the disk internals, and not only the CPU. When using low-power CPUs, not only is the disk bandwidth over-provisioned but the power consumption is also disproportionately distributed within each



(a) Fabric Node Architecture

(b) CPU-Fabric Interface

(c) I/O Cards

Figure 3. Implementing Virtualized I/O

server. Commodity disks consume between 7-25 Watts [8], which is 0.8x-3x the CPU power (8.5 W). This leads to the disk consuming a large fraction of server power, while the disk bandwidth is being under-utilized. These arguments also hold for Ethernet based network I/O [8].

4 System Design

Our production hardware based on virtualized I/O and low-power CPUs can be thought of as a **cluster-in-a-box** as it packages compute, storage, and network in a single machine. In this section we first discuss the hardware based implementation of virtualized I/O. The system architecture is then described in Section 4.2 and Section 4.3 explains the OS support needed to improve resource utilization.

4.1 Implementing Virtualized I/O

To balance resource utilization and amortize power consumption of the I/O devices, we virtualize the I/O resources. Disk virtualization allows sharing one disk among multiple CPU cores and allows configuring the ratio of CPU threads-per-disk based on application requirements. Ethernet virtualization replaces the *rack-top switch* with a low-latency, high bandwidth interconnect. This I/O virtualization is implemented in hardware and is transparent to the software.

The architectural structure enabling our implementation of I/O virtualization is an interconnect fabric that connects all the servers, disks, and Ethernet ports. This interconnect is a low-power, high-performance fabric implemented using a custom ASIC. The energy and cost overheads of the interconnect are amortized by sharing it to implement both virtualized disks and Ethernet. The ASICs implement a distributed fabric architecture in a 3D-torus topology that is able to adapt and re-route traffic around congested and failed fabric nodes. There is less than 1μsec communication delay between any two cluster nodes connected via this fabric. Each ASIC implements two fabric nodes with six links to neighboring nodes, as shown in Figure 3(a). Each link is a 2.5 Gb/sec serial link with all the system nodes and I/O cards (described later) connected in a 8x8x8 3D torus. Each fabric node connects to one CPU via a PCIe link.

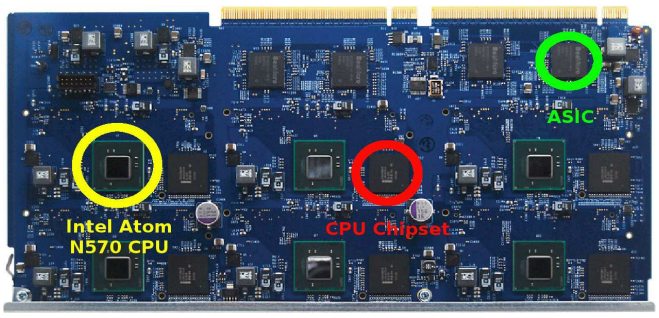
The Ethernet and disk traffic from the CPU travels over the PCIe link to reach the fabric node. Figure 3(b) shows this datapath. The node packetizes this data and routes it to the I/O cards shown in Figure 3(c). The I/O cards (E- and S-card for Ethernet and storage) terminate the fabric protocol on one end and communicate with the I/O device on the other. The power and cost overheads of these cards are trivial as they are amortized over a large number of server nodes. Virtualized disks are presented to the operating system as a standard SATA disk by the ASIC. Behind the SATA interface, the ASIC tunnels all SATA traffic through the fabric to the S-card which interfaces with the disks. Requests from many nodes can target a single disk on an S-card at the same time. The S-card then queues the requests and services them using round-robin scheduling.

Such an implementation makes the virtualization of I/O devices transparent to the operating system and eliminates any need for OS modification. With disk virtualization, the OS running on each node is presented an independent disk, which under the hood is an offset on some physical disk. As an example, if 16 CPUs are configured to share a 1 TB disk, then each CPU is presented a disk of 64 GB size. CPU-0 accesses the disk from offset 0 through 64 GB, CPU-1 accesses the same disk from offset 64 GB to 128 GB, and so on. The effect of disk virtualization is that the 1-to-1 mapping between disks and CPUs in traditional servers is altered. With our system design, one can achieve variable cores-per-disk ratio. In this work we study the impact of sharing disks among CPUs on Hadoop applications.

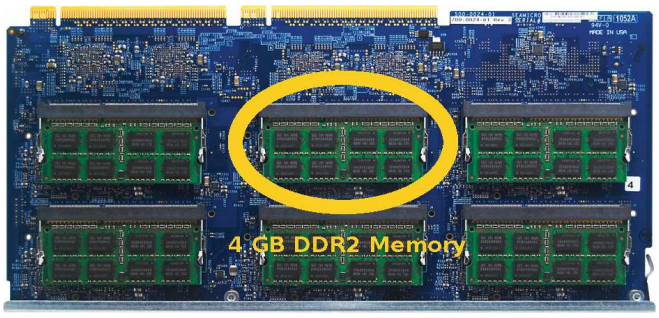
4.2 System Architecture

One of the primary concerns while designing servers using low-power CPUs is the power overheads of components like fans and power supplies [18, 9, 26]. Our system design amortizes these overheads by sharing them among a large number of cluster nodes. This is achieved by using a modular design for the server nodes and by sharing the power delivery network among all nodes. We describe these next.

A server node in the system consists of a dual core, 64-bit Atom N570 CPU and is configured with 4 GB of DDR2

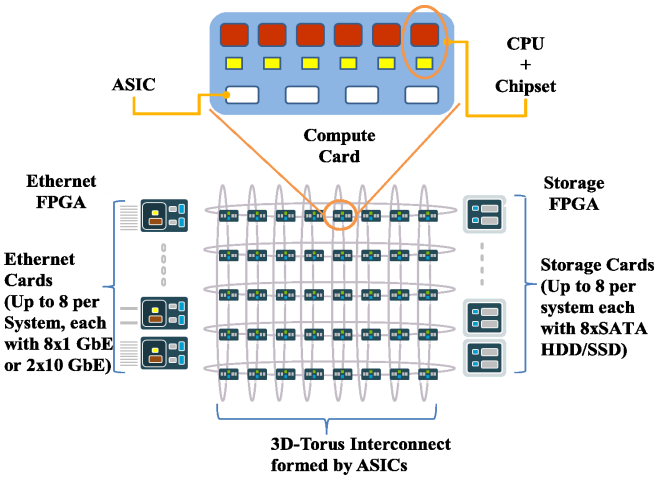


(a) C-Card Front View - Six Atom CPUs, their chipsets, and four ASICs

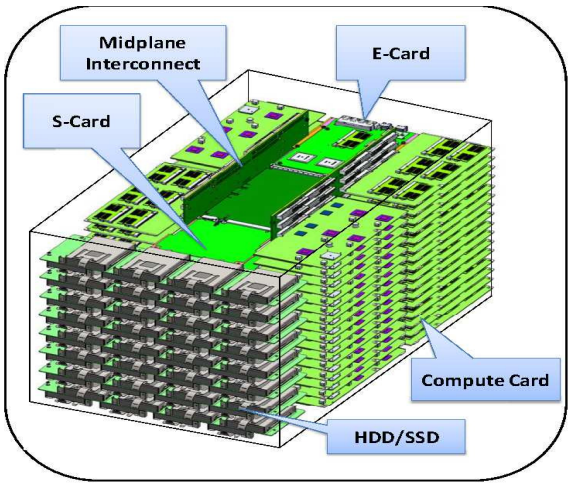


(b) C-Card Back View - Six DRAM modules, one for each node

Figure 4. Front and Back Images of a Compute-card



(a) Logical System Architecture



(b) Physical System Organization

Figure 5. System Architecture

memory. To improve power efficiency, all motherboard components except the CPU, its chipset, and the memory are removed. Each node is an independent server and runs its own OS image. There are six such nodes on a "compute card" that forms the building block of the system. A compute card for this system is shown in Figure 4. Figure 5 shows the complete logical and physical architecture of the cluster-in-a-box. The compute nodes packaged on a compute-card are plugged into the mid-plane shown in Figure 5(b) and are logically interconnected using the 3D-torus topology [12] shown in Figure 5(a). This is the same interconnect fabric described previously that enables I/O virtualization. The system supports upto 64, 1 Gbps, or 16, 10 Gbps Ethernet ports for in-bound and out-bound Ethernet traffic. These ports are implemented at the Ethernet FPGA (the E-Card). The S-Card implements the functionality to configure 64 SATA or Flash based disks per system. The complete system is packaged in a 10 Rack-Unit space including the necessary fans and 3+1 redundant power supplies. This amortizes the energy and cost overheads of these fixed components. The complete system has 384 dual-core CPUs, 1.5 TB of DRAM, and consumes less than 3.5 kW power under full load.

4.3 System Software Configuration

This section discusses the impact of various OS and Hadoop parameters on the system's resource utilization.

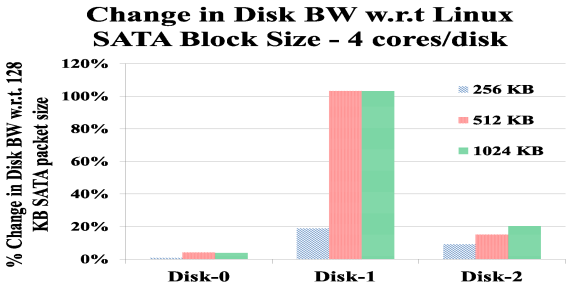
4.3.1 Linux SATA Packet Size Configuration

Virtualized disk I/O is useful in balancing resource utilization and power provisioning. However, by having to alternate between requests from different servers results in increased disk head activity. The increased head seeks degrade the aggregate disk bandwidth. For most disks this impact is negligible, but for disks from certain manufacturers the bandwidth drops significantly with virtualization. This occurs due to the poor scheduling of disk requests.

The performance of a disk depends highly on how well the scheduling algorithm can minimize the number of disk head seeks that may result from the high number of independent request streams. Scheduling occurs at several levels: 1) the operating system, 2) I/O aggregation point (S-card), and 3) controller inside the HDD. The scheduling in-

Disk	Internal Cache (MB)	RPM	Random Seek Time (ms)	Average Latency (ms)
Disk-0	32	7200	8.0	4.16
Disk-1	16	7200	11.0	4.17
Disk-2	16	7200	11.0	4.17

(a) Manufacturer Disk Specifications



(b) Impact of Linux SATA block size on raw disk bandwidth

Figure 6. Disk Characteristics

side the HDD is vendor-specific and may or may not be effective. It would be desirable to manage the disk's scheduling algorithm independently of the hardware manufacturer. In this section, we show that this can be achieved by configuring the Linux SATA driver's block size.

The Hadoop filesystem (HDFS) reads and writes data at the granularity of a block [29] and the typical HDFS block size is 64 MB or larger. This is done to minimize the number of disk head seeks when accessing data. The Linux OS on the other hand breaks any disk access command into smaller chunks that are processed by the SATA driver. The default SATA packet size for the Linux kernel is 128 KB for data reads. Thus, a 64 MB HDFS block is broken into 512 SATA packets before being issued to the disk.

With disk virtualization enabled and multiple servers sharing the same disk, we observed that the aggregate disk bandwidth for some disks reduced significantly. Breaking one large request into many small requests is not too detrimental when a disk is accessed by one CPU core. But it is detrimental when there is sharing because the disk controller's internal algorithm switches between accesses from different cores and doesn't handle all the requests from one core sequentially. We observed that the change in disk bandwidth was heavily dependent on the specifics of the disk. The request caching algorithm of the disk and the internal cache size seem to have a profound effect on raw bandwidth when virtualizing the disk.

The Table in Figure 6(a) lists disk specifications for three disks from different manufacturers. Figure 6(b) shows the impact of changing SATA packet size on aggregate disk bandwidth for these disks. Note that disk-1 shows markedly improved performance with larger packet size, while the other two show a modest increase in aggregate bandwidth (5%-20%). This shows that one way to prevent the disk from alternating between small requests from different cores is to increase the SATA packet size.

4.3.2 OS File System Configuration

Since Hadoop applications read and write a lot of data to the disk, the OS filesystem implementation becomes an important configuration parameter. Newer versions of the Linux OS typically include a journaling filesystem like the *ext4* as the default. Our experiments show that journaling filesystems perform poorly for Hadoop applications. The reason for this poor performance is the frequent updates to the journal by these filesystems. These updates induce disk head seeks which interfere with the data accesses by the application. Using a non-journaling filesystem like the *ext2* (or journal-less mode of *ext4*) improves application performance by reducing the writes caused by the filesystem.

4.3.3 Ethernet Network Driver Configuration

While experimenting with a large Hadoop cluster, we found that the network driver configuration is critical because cluster nodes communicate using the TCP/IP protocol. We found that enabling interrupt coalescing [30] and Jumbo frames [14] improved application performance. Fewer interrupts allow the CPU to do more work between interrupts, and Jumbo frames reduce the TCP/IP overheads. Jumbo frames are especially suited for Hadoop applications because large chunks of data are carried over the network during the shuffle phase, and also when the DataNodes are delivering data to remote TaskTracker nodes.

4.3.4 Hadoop Datanode Topology

We found that Hadoop application execution can be further optimized on our system by controlling the number, and placement of DataNodes in the system. Execution time can be reduced by configuring the DataNodes only on cluster nodes closer to the storage-FPGA (S-FPGA). There are three reasons for the reduction in execution time with this optimization: *(a)* Running DataNodes on only a few nodes frees up the total cluster memory used by the DataNode daemons and makes it available for the application. *(b)* There are fewer interfering requests made to the virtualized disk by multiple DataNodes [27]. *(c)* Unlike traditional clusters where network bandwidth is limited, ample fabric bandwidth provides efficient data transport among cluster nodes.

5 Results

5.1 Methodology

To quantify the impact of the described system on Hadoop MapReduce applications, we experimented using

CPU	Atom N570	# Sockets	384
Cores/Socket	2	Hyper-Threading	Yes
# Disks	64	DRAM Capacity	1.5 TB
Linux Kernel	2.6.32	Hadoop Version	0.203.0

(a) System Configuration

CPU	Core i7-920	# Sockets	17
Cores/Socket	4	Hyper-Threading	Yes
# Disks	17	DRAM Capacity	153 GB
Linux Kernel	2.6.32	Hadoop Version	0.203.0
Ethernet	1 Gbps	Ethernet Switch	Procurve 2810-24G

(b) Core i7 Cluster Configuration (Baseline)

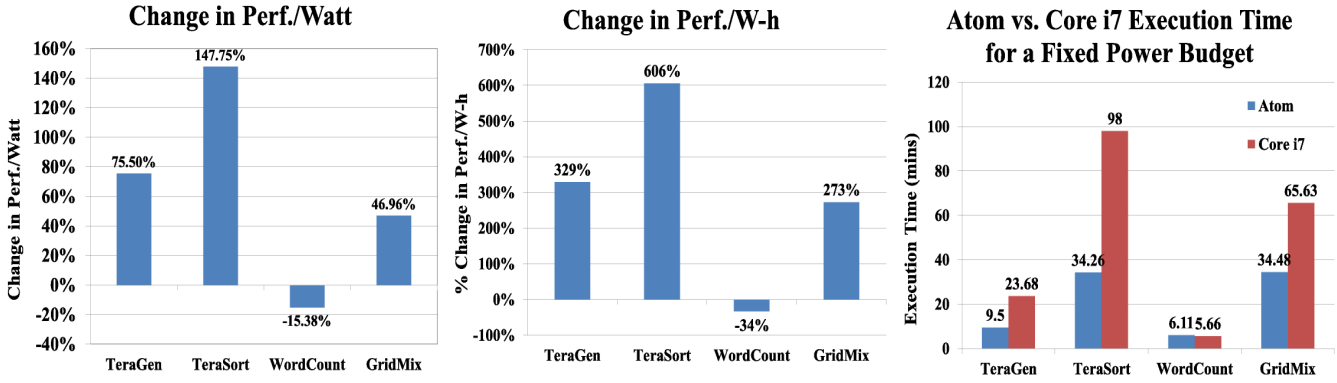
TeraGen		TeraSort		WordCount		GridMix2	
Input Dataset	NA	Input Dataset	1 TB	Input Dataset	29 GB	Input Dataset	700 GB
Output Data	1 TB	Output Data	1 TB	Output Data	5.8 GB	Output Data	220 GB

(c) Benchmark Datasets

PUE	1.45	Lifetime	3 yrs	Electric Utility Cost	$ 0.07/kWh	System List Price	$ 160,000

(d) Total Cost of Ownership (TCO) Model Parameters

Figure 7. Experimental Configuration and TCO Model Parameters



(a) (b) (c)

Figure 8. Change in Perf./Watt, Perf./W-h, and Execution Time on the System Compared to Core i7 Cluster in a Fixed Power Budget

a system with virtualized I/O whose configuration parameters are listed in Table 7(a). This system was compared to a 17-node Core i7 cluster with configuration parameters listed in Table 7(b). **The Core i7 cluster was chosen such that the two systems represent the same power envelope**. The Core i7 cluster is rated at just under 4 kW for power consumption, and our system is rated at nearly 3.5 kW. Power measurement for both systems included all the system components, specifically, it also included the Ethernet switch power for the Core i7 cluster. Table 7(c) shows the four benchmarks along with their input and output dataset sizes. These are standard benchmarks used to quantify a Hadoop clusters performance [20].

Table 7(d) shows the parameters for the total cost of ownership (TCO) model. This model is adopted from Hamilton's [6] model where: $TCO = capital\_cost + PUE{*}lifetime{*}avg\_watt{*}(\$/kWh)$ where, $avg\_watt = energy\_consumed\_by\_benchmark \div execution\_time$. PUE is the power usage efficiency of the datacenter. It measures the amount of total power consumed while delivering 1 unit of power to the servers. A PUE value of 1.5 implies that for every Watt consumed by the datacenter servers and networking equipment, 0.5 Watts are consumed by cooling, power distribution losses etc.

Throughout this section we will quantify disk virtualization using the term threads-per-disk (TPD). The total number of threads that can be executed on each cluster node is the total number of hyperthreads the socket supports. For the Atom cluster there are 4 threads/socket, and for the Core i7 cluster, there are 8 threads per socket (4 cores/socket, 2 threads/core). The number of Hadoop Map and Reduce slots that can be executed concurrently on one node of the cluster is also a configurable parameter. All experiments on our system are configured to run 2 Map and 2 Reduce tasks simultaneously, utilizing all the 4 threads on a socket. The Core i7 cluster cannot be configured to share disks among cluster nodes. To approximate the effect of disk sharing among multiple threads, we vary the number of Map and Reduce tasks executing concurrently on the Core i7 nodes. For example, with 4 Map and 4 Reduce tasks on one Core i7 node, the TPD ratio is 8. To achieve TPD of 4, the number of Map and Reduce slots is configured to 2 each.

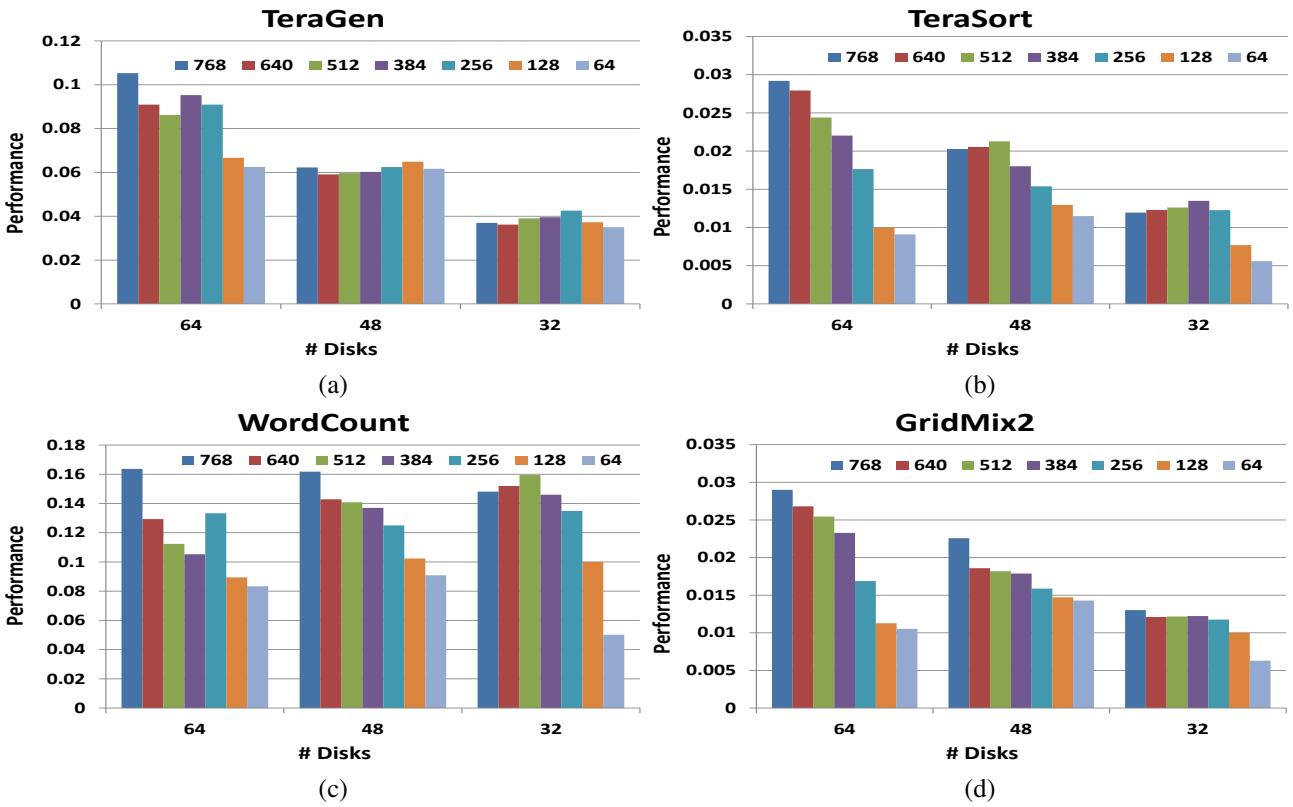


Figure 9. Performance for the System while Varying # Disks and Cores.

5.2 Evaluation

Comparison with Core i7 Based Cluster: The described system delivers higher efficiency per unit of power, and energy, while reducing execution time in the same power envelope. Due to increased power efficiency, the maximum power drawn by the datacenter can be reduced for the same performance levels and energy consumption can be reduced. Alternatively, in a fixed power envelope, performance of the datacenter can be improved by increasing the cluster size and scaling out the applications. Figure 8(a) shows the change in Perf./W of the four benchmarks when executed on the described system compared to the Core i7 based cluster. The disk and CPU intensive benchmarks TeraGen and GridMix show 75.5% and 46.9% improvement respectively, while TeraSort, which has a CPU intensive Map phase, and disk intensive Reduce phase shows a 147.8% improvement. WordCount, which stresses neither the CPU, nor the disk due to its small dataset size shows a degradation of 15.4%. Figure 8(b) shows the change in performance per unit of energy (Watt-hour) when comparing the two clusters. The trend is similar to Perf./W and shows that our system is 2X-6X more energy efficient than a commodity cluster.

Figure 8(c) compares the best execution time of the four benchmarks on our system with the Core i7 cluster. Our system outperforms the Core i7 cluster significantly, except for the WordCount application. TeraGen, TeraSort and GridMix show execution time improvement of 59.9%, 65%, and 47.5% respectively. WordCount shows a degradation of 7.9% on our system compared to the Core i7 cluster. These improvements in the same power budget come from multiple factors - increased parallelism due to the large number of servers, virtualized I/O, and lower system overheads for a server, like fans and power supplies. The performance degradation for WordCount occurs because there is not much parallelism to be extracted due to the small dataset of the application. This shows that our system is especially well suited for applications that operate on large datasets.

Analyzing the Improvements of the Virtualized I/O System: To understand the system's performance characteristics over its entire design space, we need to vary both compute and disk resources to determine the point where these resources are balanced and yield best performance. These experiments thus quantify the performance effect of adding more nodes to a cluster. In Figure 9, the performance of a benchmark for a given number of CPU cores is plotted on the Y-axis, while clustering the results based on the number of disks used on the X-axis. Each bar represents the number of cores used for the experiment. The trend to note here is that the least execution time occurs when using

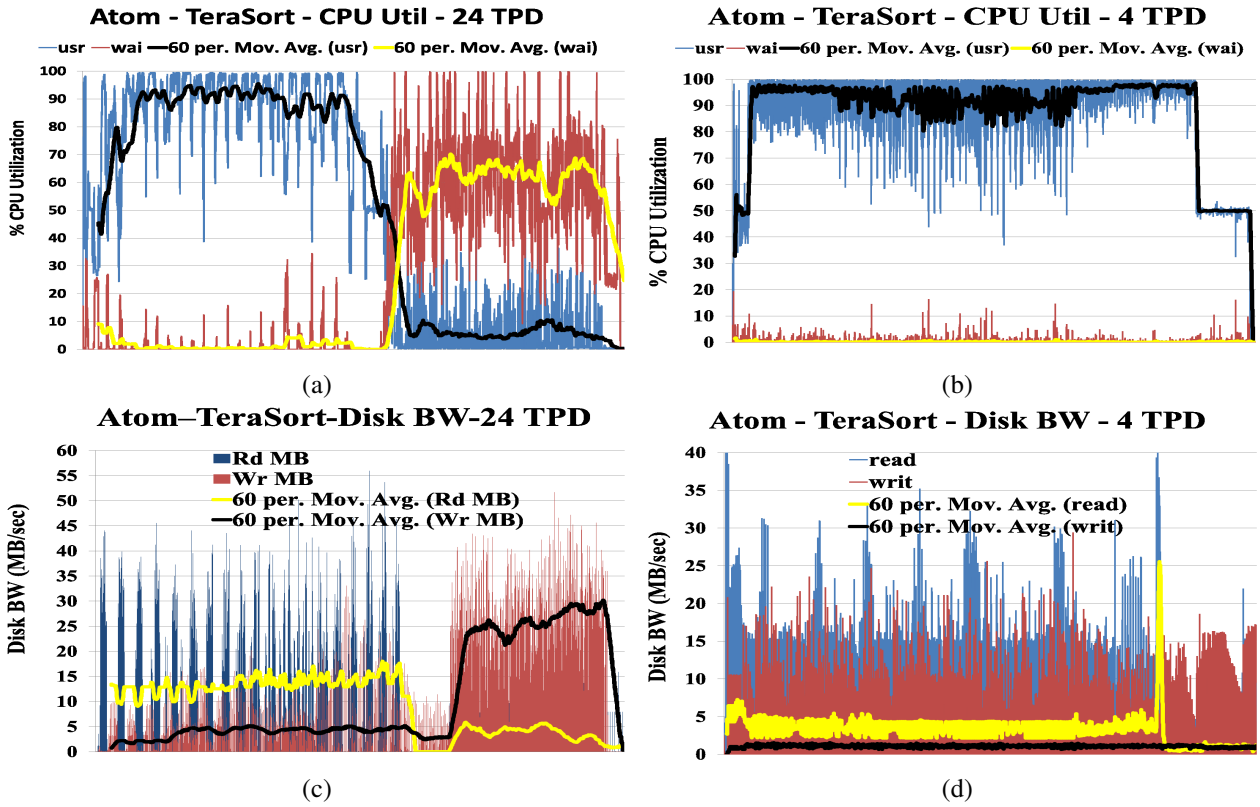


Figure 10. CPU and Disk Utilization for the Best and Worst Execution Time on the System.

the most resources (768 cores and all the 64 disks). This shows that Hadoop MapReduce jobs have improved execution time when executed on a larger cluster.

Resource Utilization: To better understand the reasons for improved performance of our system with a larger cluster size, we analyze the resource utilization of the cluster among 64d/768c and 32d/64c configurations. We observe that the overall resource utilization of the system increases when scaling to large number of cluster nodes. Figure 10 plots the disk bandwidth and CPU utilization profile for the two configurations for the TeraSort benchmark. This comparison shows that the Atom CPU can become a bottleneck when few CPUs are used for a large job. These results provide evidence that though Atom CPUs are low-power and energy efficient, it does not imply that they can be substituted for heavyweight cores for all applications and platforms. Applications that are highly parallelizable can benefit more when using large number of low-power cores.

Analyzing the Total Cost of Ownership: While scaling to a larger number of cluster nodes with our system, the provisioned system resources come in balance leading to improved power and energy efficiency. In Figure 11 we show the change in perf./TCO when the numbers of cores and disks are varied for the system. For all the benchmarks, perf./TCO is highest when using most resources, i.e., it's highest when all the 768 cores of the system are used along with all the 64 disks. For these results, the perf./TCO metric is plotted on the Z-axis while the number of CPU cores and disks is varied on the Y and X-axis respectively. The performance of a system with respect to a benchmark is defined as the inverse of the execution time of that benchmark.

Comparing Power and Energy Efficiency for the Extreme Design Points: Using more resources for traditional commodity clusters has usually come at a steep increase in energy and power consumption. Starting with a 32 disk/64 core configuration, if we were to scale the system to 384 disk/768 core, we would see a significant spike in power consumption (>1750 W assuming idle disks consume ∼7 W). Figure 12 quantifies the impact of using increased resources in a balanced manner for our system. Figure 12(a) shows that as more resources are used when moving from 32d/64c configuration to 64d/768c configuration, the improvement in Perf./W varies from 75% to 175%. Similarly, Figure 12(b) shows that the improvement in Perf./W-h for 64d/768c configuration varies from 5.3X to 14.4X compared to 32d/64c configuration. These savings come from the ability of the system to provide balanced resource utilization by virtualizing I/O, and by amortizing fixed costs like fan and power supplies.

Impact of SATA Packet Size on Execution Time As discussed in Section 4.3.1 certain disks show significant degradation in aggregate disk bandwidth when multiple

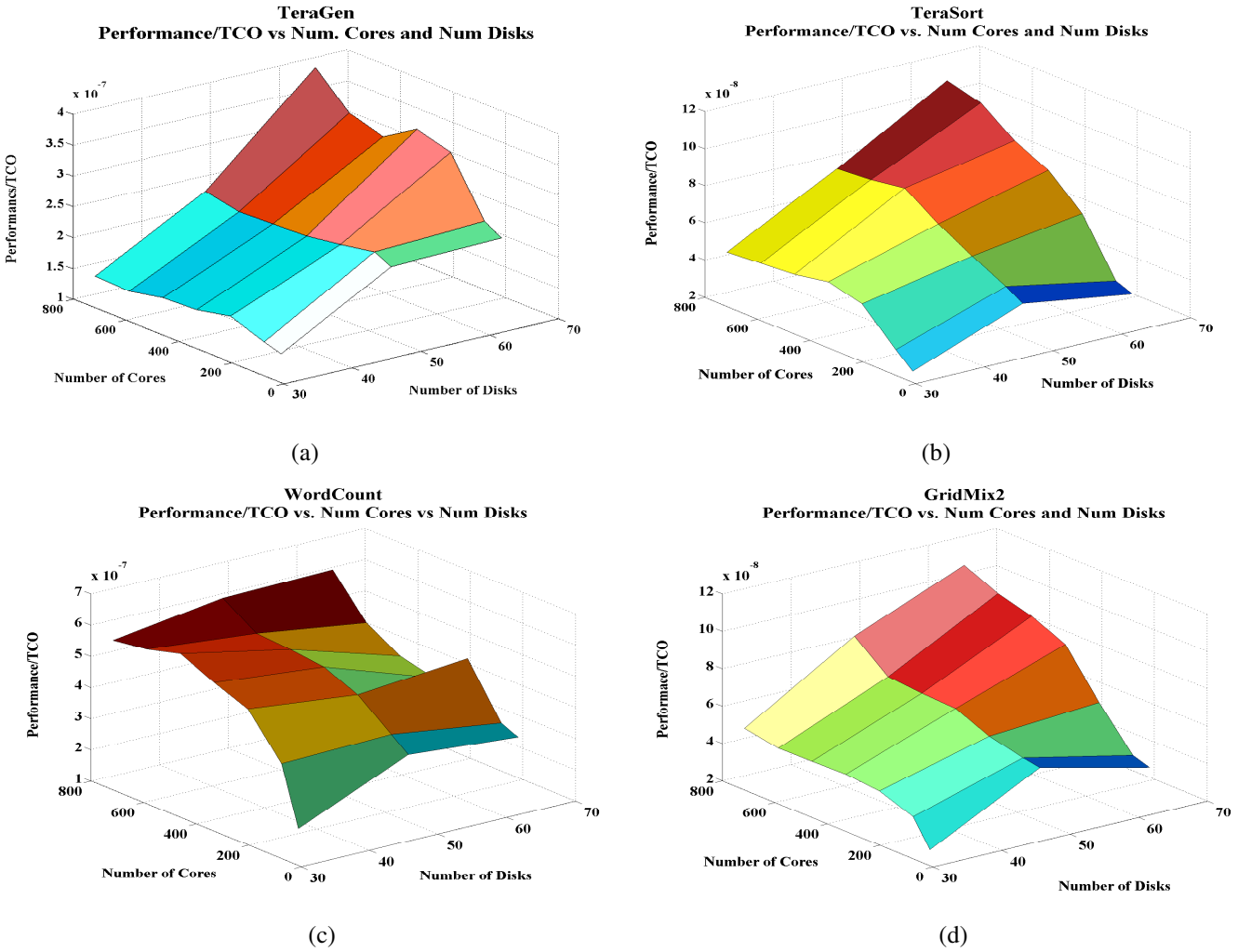


Figure 11. Performance/TCO vs. Number of Disks and Number of Cores.

cores are sharing a single physical disk. We discussed that one simple solution to increase aggregate disk bandwidth is to increase the Linux SATA driver packet size. This amortizes the cost of disk head seeks. For disks which show poor aggregate bandwidth when shared, Figure 13 shows the performance impact of increasing the Linux SATA packet size from 128 KB to 1024 KB. Figure 13(a) shows the results of these experiments for WordCount and TeraSort benchmarks with TPD ratio of 12. Figure 13(b) shows the improvement in execution time for varying TPD configurations.

6 Related Work

Optimization of compute infrastructure costs is an actively studied area. Multiple proposals advocate the use of low-power, energy efficient cores for datacenter workloads. Reddi et al. [26] quantified the cost of efficiency when running a search application on heavyweight vs. low-power cores. They show that low-power cores like the Intel Atom show improvements in power efficiency when compared to a server class CPU like the Intel Xeon. This efficiency however comes at the expense of quality-of-service guarantees due to the increase in latency and variability for search queries. They suggest over provisioning and under-utilization of Atom cores and provide evidence that even with over-provisioning, Atom based platforms incur only $0.4x$ the platform-level cost and power efficiency of the Xeon. Our scheme follows a similar chain of thought where a large number of Atom CPUs are used in the cluster. They also propose building multi-socket systems using Atom CPUs to amortize the platform level costs. This also is in line with our proposal since our scheme amortizes platform costs, but does so more efficiently by building independent compute nodes instead of building a multi-socket SMP system. Multi-socket SMP systems have higher capital costs compared to our proposal. It is an open research problem to investigate if multi-socket SMP systems deliver higher performance for scale-out architectures, or if single socket systems are better.

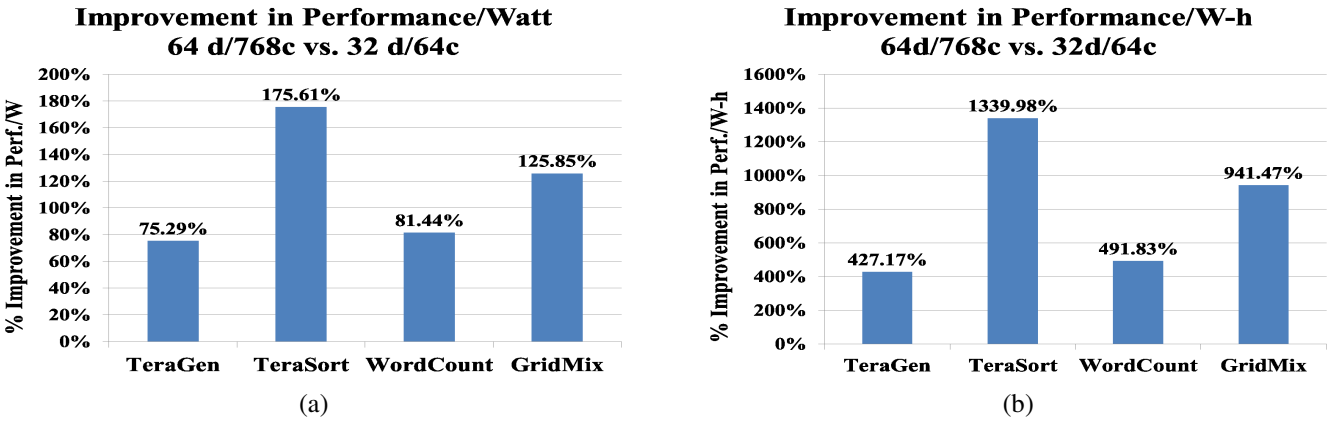


Figure 12. Power and Energy Efficiency Improvements when Adding More Resources.

Andersen et al. [9] proposed the FAWN system for datacenter key-value stores. They suggest a scheme that builds on low-power cores and Flash storage to construct an efficient high-performance key-value storage system. The proposal includes hardware-software co-design and their software innovations are primarily aimed at improving Flash storage's lifetime. The basic premise of their system is to match compute and I/O abilities for a specialized task of key-value datastore. Our system however is suited to general computational requirements of modern datacenters, while FAWN is targeted at a specialized application. The main ideas in our work are orthogonal to those in the FAWN architecture.

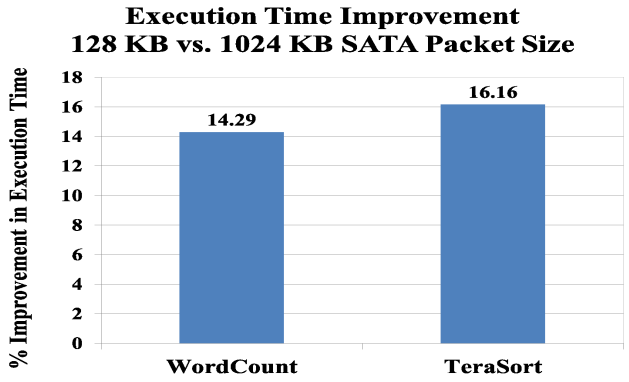
Hamilton [18] points out that servers are out-of-balance in terms of allocated resources, and raw server performance is a wrong metric to evaluate systems for web-scale workloads. Based on these observations, his CEMS proposal utilizes low-power CPUs that are optimized for work done per Dollar. CEMS however is still designed around traditional server architectures which do not amortize the cost of components like fans, and power supplies. Apart from amortizing these costs, with virtualized I/O our system provides further improvements due to better resource balance.

Besides this large body of work that advocates using low-power CPUs for web-scale workloads, there is work that conversely argues for using heavyweight cores for web-scale workloads. Hölzle [19] argues that heavyweight cores with high single thread performance are desirable for Internet workloads compared to low-power cores with lower single thread performance. He argues that once single thread performance of low-power cores lags by a factor of two or more behind heavyweight cores, it becomes harder to improve cost effectiveness of low-power cores. This occurs due to multiple factors like higher system overheads (like cabling, enclosures) and power inefficiency of components like disks and network. Our design aims at precisely this problem and solves it using virtualized I/O and sharing fixed components.

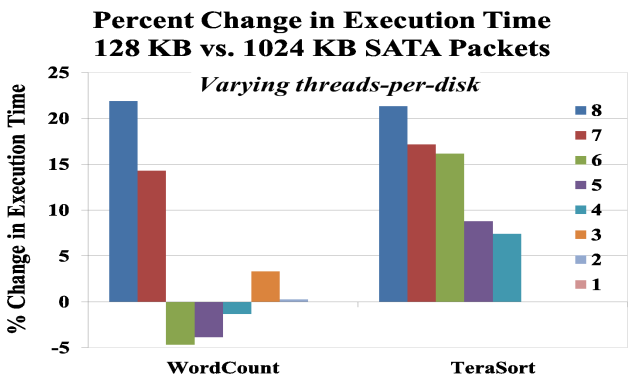
Meisner et al.[25] performed a study to determine if low-power server designs imply energy efficiency for datacenters. Based on highly tuned server designs from the past six years, they conclude that high-power designs are not energy inefficient when used to build large datacenters. We agree with these arguments, but our novel design, which amortizes system component costs and energy by sharing resources, is able to provide more headroom in achieving cost effectiveness. Lang et al.[22] show that a system built using low-power CPUs shows disproportionate scale-up characteristics, making such systems an expensive and lower performance solution. Our system addresses this exact problem by amortizing server costs and providing resource balance.

7 Conclusions

The challenges in building energy efficient clusters of servers with low-power CPUs are two fold: *(1)* balancing I/O resource utilization with power consumption, and *(2)* amortizing the power consumption of fixed server components like fans and power supplies. In this work we show that the I/O resource utilization and power consumption can be balanced by using a virtualized I/O approach. The fixed component power overheads on the other hand can be amortized by sharing them among many servers with low-power CPUs. We demonstrate a system built on these ideas and study its characteristics for web-class applications. The system shows significant improvements in Perf./W and Perf./W-h for Hadoop applications and scales to a large number of cluster nodes. Compared to a commodity cluster using heavyweight CPUs in the same power budget, our system delivers 3X average improvement in Perf./W-h, and 64% average improvement in Perf./W. For a fixed power budget, the execution time also improves by 57% compared to a traditional cluster for benchmarks that showed improved performance.



(a) Percentage improvement in execution time with 1024 KB SATA read packet compared to 128 KB SATA packet, with 12 TPD



(b) Percentage improvement in execution time with 1024 KB SATA read packet compared to 128 KB SATA packet, with variable TPD

Figure 13. Impact of Changing SATA packet Size on Performance

References

[1] Amazon Web Services. http://aws.amazon.com/.

[2] Hadoop at Yahoo! http://developer.yahoo.com/hadoop/.

[3] Hadoop Distributed Filesystem Architecture Guide. http://hadoop.apache.org/common/docs/current/hdfs_design.html.

[4] Intel Atom Processor N570 Specifications. http://ark.intel.com/products/55637.

[5] Intel Xeon Processor E7-8870 Specifications. http://ark.intel.com/products/53580.

[6] Overall Data Center Costs. http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx.

[7] The Apache Hadoop Project. http://hadoop.apache.org/.

[8] Internet-Scale Datacenter Economics: Costs and Opportunities. In *High Performance Transaction Systems*, 2011. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_HPTS2011.pdf.

[9] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of SOSP*, 2009.

[10] O. Azizi, A. Mahesri, B. Lee, S. Patel, and M. Horowitz. Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. In *Proceedings of ISCA*, 2010.

[11] D. Borthakur, J. Gray, J. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache Hadoop Goes Realtime at Facebook. In *Proceedings of SIGMOD*, 2011.

[12] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 1st edition, 2003.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of OSDI*, 2004.

[14] P. Dykstra. Gigabit Ethernet Jumbo Frames: And Why You Should Care. http://sd.wareonearth.com/ phil/jumbo.html.

[15] X. Fan, W. Weber, and L. Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of ISCA*, 2007.

[16] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *Proceedings of SOSP*, 2003.

[17] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *Proceedings of ISCA*, 2011.

[18] J. Hamilton. Cooperative Expendable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2009.

[19] U. Hölzle. Brawny Cores Still Beat Wimpy Cores, Most of the Time. In *IEEE Micro*, 2010.

[20] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench Benchmark Suite : Characterization of the MapReduce-Based Data Analysis. In *International Conference on Data Engineering Workshops*, 2010.

[21] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server Engineering Insights For Large-Scale Online Services. In *IEEE Micro*, 2010.

[22] W. Lang, J. Patel, and S. Shankar. Wimpy Node Clusters: What About Non-Wimpy Workloads? In *Proceedings of the Sixth International Workshop on Data Management on New Hardware*, 2010.

[23] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. Reinhardt, and T. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of ISCA*, 2009.

[24] K. Lim et al. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *Proceedings of ISCA*, 2008.

[25] D. Meisner and T. Wenisch. Does Low-Power Design Imply Energy Efficiency for Data Centers? In *Proceedings of ISLPED*, ISLPED, 2011.

[26] V. J. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. In *Proceedings of ISCA*, 2010.

[27] J. Shafer, S. Rixner, and A. Cox. The Hadoop Distributed Filesystem: Balancing Portability and Performance. In *Proceedings of ISPASS*, 2010.

[28] U.S. Environmental Protection Agency - Energy Star Program. *Report To Congress on Server and Data Center Energy Efficiency - Public Law 109-431*, 2007.

[29] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[30] M. Zec, M. Mikuc, and M. Zagar. Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput. In *Proceedings of SoftCOM*, 2002.