Energy Efficient Scale-In Clusters with In-Storage Processing for Big-Data Analytics

I. Stephen Choi and Yang-Suk Kee
{stephen.ch, yangseok.ki}@samsung.com
Memory Solutions Lab.
Samsung Semiconductor Inc., San Jose, CA 95134

ABSTRACT

Big data drives a computing paradigm shift. Due to enormous data volumes, data-intensive programming frameworks are pervasive and scale-out clusters are widespread. As a result, data-movement energy dominates overall energy consumption and this will get worse with a technology scaling. We propose scale-in clusters with In-Storage Processing (ISP) devices that would enable energy efficient computing for big-data analytics. ISP devices eliminate/reduce data movements towards CPUs and execute tasks more energy-efficiently. Thus, with energy efficient computing near data and higher throughput enabled, clusters with ISP can achieve more than quadruple energy efficiency with fewer number of nodes as compared to the energy efficiency of similarly performing its counter-part scale-out clusters.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Sytles–*Heterogeneous (hybrid) systems*; D.1.3 [**Software**]: Concurrent Programming–*Parallel programming, Distributed programming*

1. INTRODUCTION

Big-data analytics challenge traditional computing platforms. Exa-/zetta-byte of data is generated and analyzing such enormous amount of data drives radical changes. Researchers in both academia and industry have been working on building novel computing platforms to cope with the large-scale data. Consequently, large clusters are omnipresent to analyze the data by adding more commodity servers, i.e. *scale-out* clusters, for scalability and cost-effectiveness. On the other hand, data-intensive programming frameworks, e.g. MapReduce or Dryad, are proposed and largely deployed in data-centers.

While such programming and execution frameworks provide a single view of data with cost-effectiveness and great scalability, data communications via file based on distributed file systems introduce heavy I/Os. As a result, memory and storage bandwidths can be limiting constraints in achieving higher performance. Unfortunately, continued core scaling broadens the gap between computing power and memory [24] and storage bandwidths. Therefore, the scale-out approach will be used to reduce bandwidth requirement per node while under-utilizing other resources, such as CPUs, due the bandwidth limits.

Although the scale-out approach can address such bandwidth-wall problems, it introduces a more fundamental challenge: energy consumption of data movements increases due to the larger size of clusters. This problem will only get worse with technology scaling because data-movement energy consumption does not scale as well as computational energy consumption. As a result, scaling out clusters will be prohibitively expensive due to the energy inefficiency: data movement consumes more energy than its associated computation.

This energy inefficiency problem can be alleviated by reducing/removing data movements. We consider In-Storage Processing (ISP) for this purpose. With ISP-enabled storage devices, *scale-in* clusters, i.e. populating fewer number of nodes, are expected to outperform the energy efficiency of existing scale-out clusters. Such improvement mainly comes from alleviating two energy inefficiency sources: (1) data movements, (2) computation. Costly data movements can be significantly suppressed once the locality–processing data at the storage device in this context– is effectively utilized. That is, due to the close distance from storage media, ISP device can realize lower energy consumption for data movement compared to moving data toward CPUs. More importantly, scale-in clusters can alleviate the energy consumption of remote data accesses due to the shorter interconnects between nodes by enabling scaling in, *lower-degree of scaling out*. Second, energy-efficient job execution is achievable with ISP device's low-power designs by either eliminating overheads in general-purpose CPUs or adding special-purpose logics.

In this paper, we show that ISP can be a key enabler to achieve high energy efficiency for data-intensive applications. By offloading bandwidth-consuming computation inside ISP storage devices, both memory bandwidth and storage bandwidth requirement per node are relaxed and thus enable more tasks runnable per node. As a result, scale-in approach can be adopted to run the same job to achieve the similar performance at much lower energy consumption.

MEMSYS '15, October 05-08, 2015, Washington DC, DC, USA

DOI: http://dx.doi.org/10.1145/2818950.2818983

The main contributions of this paper are as follows:

- We study big-data analytics from a computer architecture perspective and show that memory and storage bandwidths are the main bottlenecks in clusters with commodity servers.
- We conduct a limit study to understand potential performance improvement with high-bandwidth storage devices.
- We evaluate the energy cost/inefficiency of scale-out clusters with projected technology scaling. We show that with 7nm process technology, data-movement energy consumption takes around 85% of total energy consumption while the computation energy accounts for only 15%.
- We introduce scale-in clusters with ISP for big-data analytics. This approach takes advantage of reduced local and remote data movements along with more efficient computation.
- We show that scale-in clusters with ISP can improve the overall energy efficiency up to 5.5X according to our model-based evaluation.

2. BACKGROUND AND MOTIVATION

2.1 Big Data and I/O

Coordinating software tools and architectures is common in high-performance computing, e.g. MPI or OpenMP and specially designed clusters or large-scale shared-memory machines. However, in the cluster computing for data-intensive applications, programming and execution frameworks provide a single view of data while working with commodity servers. Virtually all of these frameworks heavily rely on data parallelism and have been built for large-scale parallel executions [3]. Some widely used frameworks are MapReduce [9] and Dryad [13]. Such frameworks come with important commonalities in their parallelism granularity. Each *job*, computational work, can be submitted to a framework and a job is defined as a DAG (Directed Acyclic Graph) of *phases*, where each phase consists of many fine-grain *tasks* utilizing data parallelism. The input and output of each phase/task consists of one or more blocks of file and tasks of a phase have no dependencies among them.

Spark is recently introduced and is compatible with Hadoop– an open source implementation of MapReduce– while providing up to 100X speedup compared to Hadoop by implementing in-memory computing [33]. Spark divides input data into many chunks, *splits*, and each worker executes jobs on associated splits for parallelization. This approach is particularly effective not only for exploiting the cost-effectiveness of commodity servers, but for parallel execution of distributed data to exploit its data locality. However, this involves explicit data movements via files for its communications, utilizing distributed file systems oftentimes.

One of the most important characteristics from the architectural perspective is: communications between tasks/phases and data synchronizations are performed by utilizing file systems. That is, many parallel tasks read and write data for communications between them. Moreover, data should be committed to storage for its synchronization. As a result, MapReduce-like frameworks introduce a new challenge of heavy I/Os that might cause I/O bottlenecks in commodity servers despite of its cost-effectiveness (details of data movements in MapReduce-like frameworks will be discussed in Section 4.2.)

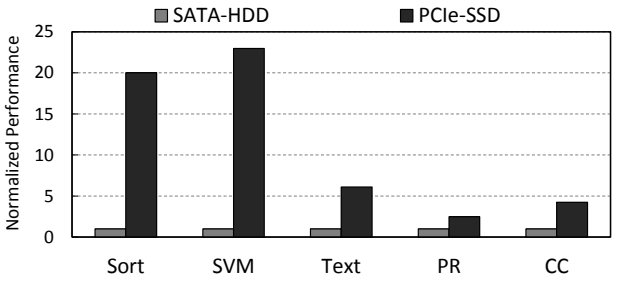

Figure 1: Performance gain with data-intensive applications on Spark framework by using higher-bandwidth storage devices.

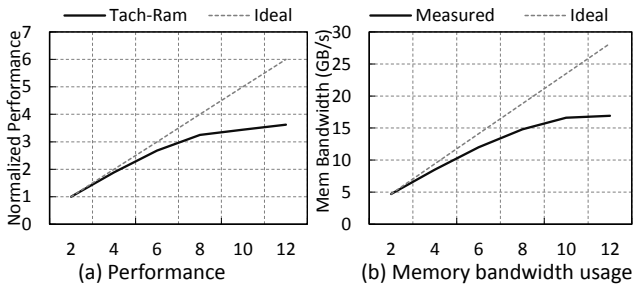

Figure 2: Performance scaling by adding more cores per data node with PageRank and measured memory bandwidth usage per node.

2.2 Bandwidth Limits

Due to such heavy I/Os, the bandwidth of storage device becomes a limiting factor for conventional systems, especially when the page cache performs poorly (e.g. data nodes commonly serve HDFS and/or other distributed DB causing page-cache interferences). Figure 1 shows a limit study result of performance gain on Spark clusters by replacing the storage device. We evaluate the performance of five workloads with SATA HDD and PCIe SSD (see Section 4.1 for the details). As shown, higher bandwidth of the storage device can greatly improve the overall performance. For example, clusters with PCIe SSDs can improve the overall performance of SVM as much as 23X compared to clusters performance using HDDs. On average, higher performance storage devices improve the performance by 11X. This is a striking counterexample to the general conception that most application are CPU bound [21].

Moreover, memory bandwidth limits performance even after resolving the limited storage bandwidth problem. Figure 2-(a) shows performance scaling by adding more cores on the same clusters while running PageRank. We evaluate the performance of PageRank from 2 cores to 12 cores per data

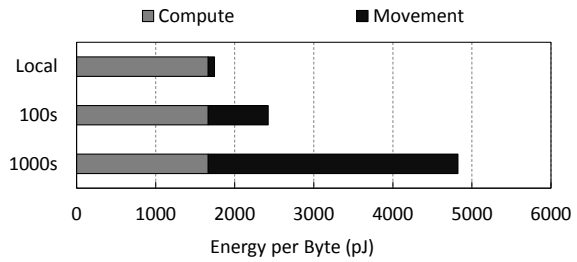

Figure 3: Data-movement and its computation energies per Byte with different sizes of clusters. Scale-out clusters introduce high data-movement energy consumption as cluster size increases.

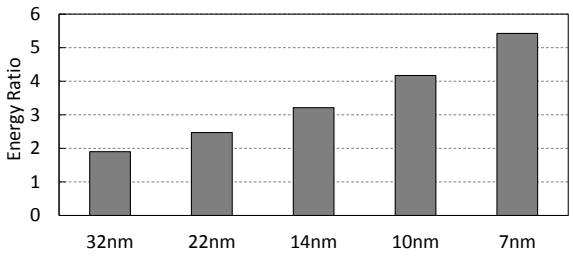

Figure 4: Energy consumption ratios, data-movement energy to its computation energy per Byte, in scale-out clusters that comprises thousands of nodes with process technology scaling.

node. Adding more cores does not scale well and 12-core configuration achieves only 60% of the performance with ideal scaling. On the other hand, Figure 2-(b) shows memory bandwidth usage. Memory bandwidth does not scale well either and its poor scaling explains the performance scaling of Figure 2-(a).

To resolve such performance bottlenecks caused by limited bandwidths, one can add more servers, i.e. scale-out approach, to distribute the bandwidth usage, resulting in the lower bandwidth requirement per data node. As a result, higher performance can be achieved.

2.3 Energy Cost with Scale-out Cluster

Despite the performance improvement, scale-out clusters increase the energy consumption of data movements due to its longer global interconnects as adding more nodes. More specifically, scaling-out generate more remote data movements of ①, and ⑦ in Figure 6. Moreover, it also causes more data movements in *shuffle*, ④. As a cluster size grows to cope with the ever-increasing data, the distance between nodes increases and its energy consumption is a power function of the distance [5]. As a result, data-movement energy easily becomes a dominant factor in the overall energy efficiency of scale-out clusters. As such, it is important to understand the energy consumption of data movements in scale-out clusters compared to its computation energy.

To compare the energy consumptions of data movements to its computation energy, we use *instruction per byte* (IPB) of representative data-intensive applications [7]. Based on the average IPB, shown in Table 1, and the computation energy per instruction, shown in Table 2, we evaluate the computation energy per Byte. On the other hand, for the data-movement energy, we use data movement energy for global interconnect made of metal wires in Table 2 (details will be discussed in Section 4.2).

Comparing the data-movement energy to its computation energy, scaling clusters out increases the data-movement energy significantly. Figure 3 shows the increased energy by comparing single-node (local) execution, clusters with hundreds of nodes, and clusters with thousands of nodes. Although the actual number of instructions, the size of input data, and data-movement distances vary across ISAs, data formats, programming model, execution frameworks, and cluster configurations, the key observation is vivid: the energy consumption of data movements will dominate the total energy consumption as the number of nodes increases. For example, in clusters that comprise thousands of nodes, data-movement energy consumption is almost double the computation energy. This is a dramatic change considering that the data-movement energy accounts for less than 5% of its computation energy in a single-node computing platform. Thus, adding more commodity-server-based nodes might not be viable beyond a certain point due to the longer interconnects caused by the increased physical dimension of the system.

Unfortunately, this problem will only get worse as technology scaling continues. Interconnect energy will not be reduced as fast as computation energy will be. 1.6X energy reduction of interconnects will be expected in coming five process generations, *i.e.* scaling down to 7nm, compared to 6X reduction in computation [5] as shown in Table 2. Based on this interconnect energy projection, Figure 4 shows expected ratios of data-movement energy to its computation energy in scale-out clusters that comprise thousands of nodes with different process technologies. At 7nm, the data-movement energy will be more than five times of its computation energy, and thus will dominate the overall energy consumption. This is very inefficient energy usage for computing, hence suppressing data movements to reduce its energy consumption will be crucial to achieve energy efficient computing in the foreseeable future.

2.4 In-Storage Processing

To suppress data movements, we consider moving computation towards the place where data reside, i.e. in-storage processing. The key objective of ISP is offloading bandwidth-consuming tasks to storage devices so that CPUs can execute additional tasks. Although ISP is not a new idea, its concept had been around for decades [1], its necessity and effectiveness in data-intensive applications is resurging as SSDs get pervasively deployed. Note that SSDs' superior performance, energy efficiency, and energy proportionality [29] over HDDs have catalyzed broad and fast adoption. Such prevailing SSD preference is greatly aligned with the exploration toward ISP. In fact, there already is a handful of related studies and Section 6 summaries these.

3. SCALE-IN CLUSTERS WITH ISP

The benefit of ISP can be summarized in three ways. First, the source of energy inefficiency in the local data movements

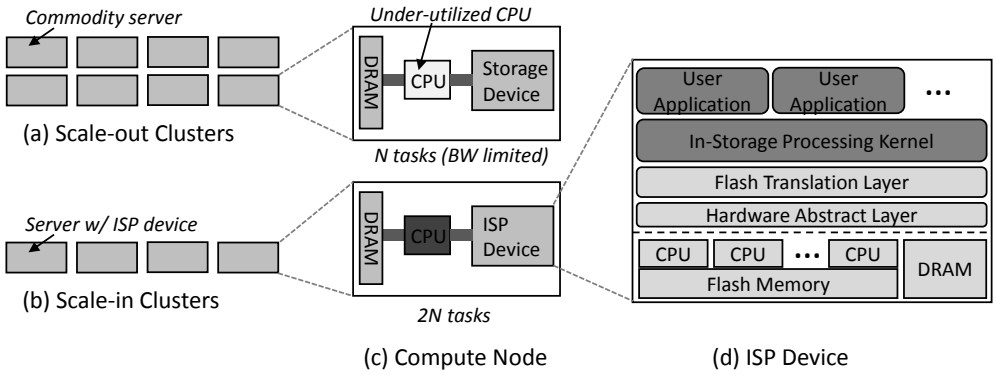

Figure 5: Scale-in clusters with ISP. ISP devices can be implemented utilizing modern SSD architectures.

can be squeezed out by computing near at the data. Among the data movements in MapReduce-like programming and execution model, the data movements between CPUs and local storages can be effectively removed by processing data at the storage. Even with partial computation in storage devices, it can effectively reduce I/O in the four local data movements in Figure 6 because some operations reduce the output data footprint, *e.g.* filter or preprocessing in general, and specifically such as *filter(), sample(), count(), collect(), or lookup()* operations in Spark. Second, ISP devices can execute tasks more energy efficiently. Embedded processors can benefit from its low-power process technology, lower overhead instructions, more efficient VLIW/SIMD, and application-specific instructions [12]. Third, each node can execute more tasks due to the computation offloading to ISP devices. The increased throughput per node enables reduced size of clusters so that a large amount of energy inefficiency from remote data movements can be squeezed out.

Due to the benefits of ISP for energy-efficient computing for data-intensive applications, we propose scale-in clusters with ISP devices. As shown in Figure 5, the same performance is achievable with lower degree of scale-out, i.e. scale in, by executing more tasks per node. Alleviating storage and memory bandwidth bottlenecks enables each node to run more tasks as shown in Figure 5-(c). Given this potential benefits, implementing ISP devices is the key challenge.

While modern SSDs have enough hardware components, such as multi-processors and DRAMs to implement ISP, developing a heterogeneous system that seamlessly work with programming and execution frameworks is a challenging task. In fact, the underlying hardware components inside SSDs shown in Figure 5-(d) are getting richer to cope with increasing demands for higher performance, *i.e.* higher IOPS, sustained data transfer, shorter response time and better durability. As a result, the transformation from an SSD into an ISP device require more work on SSD firmwares, host interfaces and communication protocols, and programming models.

For example, ISP firmware should include an ISP kernel to bridge device-side applications and the lower layers including flash translation layer (FTL) and hardware abstract layer (HAL). Meanwhile, a programming model should address (a)synchronous communication mechanisms between host CPUs and task granularities. An ISP development environment must provides integrated cross compilation both for a host binary and a device binary. An ISP runtime environment needs to implement task scheduling and load balancing. On the other hand, standardization is necessary to provide a unified coherent memory space such as CAPI [27] and an ISP host controller interface specification.

4. EXPERIMENTAL METHODOLOGY

In this paper, there are two parts of evaluations. We evaluate bandwidth limits in data-intensive applications using Spark clusters. Then we evaluate scale-in ISP cluster energy efficiency gain using computation and data-movement energy models.

4.1 Spark Clusters

4.1.1 Cluster Configuration

We use commodity cluster system consists of 8 data nodes and 1 name node. Each node is a Dell Precision T7600 server with two 6-core Intel Xeon E5-2630 CPUs (12 cores per node). Hyper-threading and Intel Turbo Boost are disabled for consistent results. Each node comprises 64GB DDR3-1600 memory, three 2TB SATA HDDs, 250GB Dell SATA SSD, and 512GB PCIe SSD (Samsung XP941). The OS uses the Dell SSD. Two HDDs are used for HDFS and another HDD is used for the RDD caching device. All nodes are connected via 10 Gigabit Ethernet using Intel X520 NICs so that the network bandwidth does not limit the overall performance.

4.1.2 Spark Framework

We use Ubuntu 12.04 LTS, OpenJDK 1.6.0_27, Spark 1.0.1, Tachyon 0.4.1, and Hadoop 1.0.4 for HDFS. In this study, we first tune and optimize Spark for good baseline performance. Such tuning and optimization include JVM tuning, specifically garbage collection, *GC*, and number of JVM instances, and a memory caching scheme selection. We use *parallel old GC* with -XX:UseParallelOldGC flag. We use Tachyon [18] to evaluate in-memory caching and shuffling while keep GC time low by avoiding placing data in JVM heap space which generates substantial GC times. Finally, we use a JVM instance per core to reduce multi-Java threads overheads.

4.1.3 Workloads and Datasets

We use 5 representative Spark applications to evaluate storage bandwidth limits. We use ported TeraSort – originally written for Hadoop – (Sort), Support Vector Machine (SVM), in-memory text search (Text), PageRank (PR), and Connected Component (CC).

We generate synthetic input data of 100G size for TeraSort using TeraGen. We use *mnist8m* dataset, size of 12G and 8.1 million data of 784 features [6], for SVM, and Twitter's social graph dataset of 22G size consists of 41.7 million users profiles, 1.47 billion social relations [17] for PageRank and Connected Component. We use Amazon Movie Reviews of 100G, generated from the model of 8 million reviews [30], for in-memory text search.

4.1.4 Storage Bandwidth

We configure Spark so that RDD caching and shuffling data is written to a storage device. Spark provides an API for this and we use StorageLevel.DISK_ONLY for RDD caching in a storage device. The measured storage bandwidths of SATA HDD and PCIe SSD are around 162/100 MB/s and 1204/1096 MB/s (r/w), respectively (we use fio with 128K block size, 32 IO depth, and 12 concurrent jobs considering number of cores per node).

For a limit study purpose, we assume poor page caching environment by running multiple services on data nodes. For this purpose, we drop the page cache every second.

4.1.5 Memory Bandwidth

We use Tachyon to place RDDs in memory and utilize *ramfs* to keep shuffle data in memory at the same time. Note that the number of job iteration is limited due to the increasing size of shuffle data while the physical memory capacity is constant. We control the parameters not to cause page swap happening which is beyond the scope of this work. We use performance counters of CPUs to measure memory bandwidth [10]. The reported numbers are the average number of gathered information across the data nodes.

4.2 Model-Based Scale-In Clusters with ISP

We evaluate scale-in ISP clusters from the data perspective. We first pick data-intensive applications, then calculate the number of instructions per input data in Byte. This provides a simple way to evaluate energy needed not only for compute, but also movement per Byte. Combining such IPB with data flow model described in Figure 6, we evaluate the total energy consumed in different cluster settings varying its size and data movement ratios.

4.2.1 Workloads

We use data-intensive applications from [7] and use average IPB for our evaluation.

4.2.2 Data Movements

We model the data movements in MapReduce programming and execution framework [9]. The data movements in Spark is straight forward to extend from here. As shown in Figure 6, *map* or *reduce* phase accesses its local storage devices (data movements of ②, ③, ⑤, and ⑥ in this figure). Spark utilizes DRAM memory to store intermediate data, but still accesses storage devices for caching RDDs or for RDD check pointing [33] and keeping all RDDs or shuffle data in DRAM is commonly prohibited due to the limited DRAM capacities[1] As a result, all fine-grain tasks read and write data from/to storage devices at its beginning and end.

	IPB		IPB
scalParC	133.7	hop	41.2
k-means	117.1	linear_regression	40.2
word_count	87.1	histogram	37.4
bayesian	83.6	grep	4.6
string_match	54		
		average	66.5

Table 1: Instruction per Byte of data-intensive applications [7].

	32nm	22nm	14nm	10nm	7nm
compute	25	17.5	12.2	8.5	6.0
local I/O (1m)	40	36.4	33.1	30.1	27.3
remote I/O (10m)	200	181.8	165.3	150.3	136.6
remote I/O (100m)	1000	909.1	826.4	751.3	683.0

Table 2: Energy consumption of data computation, local data I/O, and remote data I/O, per Byte (pJ) [5].

We assume two local data movements, from storage devices to CPUs and vice versa, in single-node computing and four local and three remote data movements in MapReduce-like computing platforms as shown in Figure 6 (minimum two phases and one remote accesses and two local accesses per phase, plus one final write as remote access). Iterative jobs [2] can be also considered in the same way by adding three data movements per additional iteration: data movements of ③, ④, and ⑤ in this figure.

4.2.3 Data Movement and Computation Energies

We use two energy scaling projections: data movement and computation. We use the model presented in the recent work [5] for both. Five different technology numbers for compute energy are used, from 32nm to 7nm in this study. To model the data-movement energies in clusters, we use three different global interconnect energies for local I/O, remote I/O for clusters with 100s nodes, and remote I/O for clusters with 1000s nodes. The underlying assumption in comparing compute over data movement is the compute energy reduces by 6x, whereas the global interconnect energy reduces by only 60% [5].

To distinguish local data accesses and remote data accesses over networks in modeling of electrical signaling energies, we selected three representative interconnect energies according to distances, <1M, ˜10M, and ˜100M, as shown in Table 2 (32nm parameters are used for this comparison).

[1]Storage class memory, possibly will provide much larger space with PCM or STT-RAM, than DRAM, is beyond the scope of this article.

[2]Many big data analytics are iterative, *e.g.* machine learning. Spark is highly focused on iterative jobs and achieves 100x performance gain over Hadoop in some cases [33]

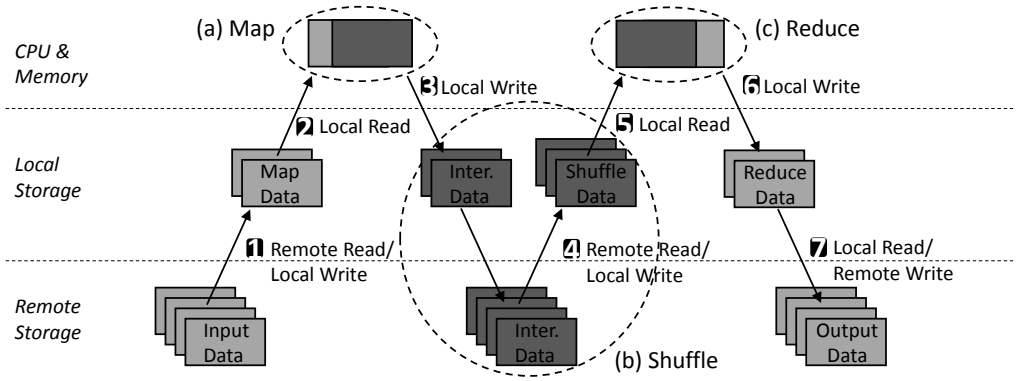

Figure 6: Conceptual data movements in MapReduce framework assuming a simple non-iterative case. Iterative computation will add three data movements for shuffling, *i.e.* 3,4 and 5, per iteration.

4.2.4 *Efficient Computation at ISP device*

We assume 25X better energy efficiency with ISP computation compared to CPU execution. We believe 25X better energy efficiency is reasonable considering the energy efficiency gain spectrum up to 500X, and individual gains of 7X with wider SIMD, 10X with flexible instruction and data fetch [12], 10X with lower-frequency process technology for the better energy efficiency with embedded processors [32].

5. ENERGY EFFICIENCY WITH ISP AND SCALING IN

With scale-out clusters, a solution to reduce the bandwidth requirement per node caused by heavy I/Os, the overall data movements increase. As such, the energy consumption for data movements should be reconsidered to understand its significance along with potential energy inefficiency of the movements. We evaluate the potential energy efficiency gain of scale-in ISP clusters by considering perspectives of local data movement, efficiency of computation, and remote data movements. Our approach can address the following problems found in conventional systems. First, all data must move from storage to CPUs for its execution and the processed data must also be written back to the storage, causing another data movement. Second, CPUs could be inferior in its energy efficiency compared to simpler cores because modern general-purpose CPUs waste large amount energy to support out-of-order execution and speculative execution to exploit the instruction-level parallelism (ILP) [12]. Third, the energy inefficiency of remote data movements is mainly caused by scaling out.

We believe that scale-in clusters with heterogeneous architectures utilizing ISP would greatly improve the energy efficiency by eliminating such inefficiencies of scale-out clusters with commodity servers. To evaluate the potential energy efficiency improvement, we use models described in Section 4.2.

5.1 Evaluation

A heterogeneous node with ISP device(s) can execute more tasks not only by executing tasks both at CPUs and ISP devices, but also by alleviating storage bandwidth bottleneck due to its squeezed-out local data movements. Thus, clusters that consist of heterogeneous nodes with ISP devices can maintain a lower degree of scale-out. The benefit of scale-in clusters is straightforward: shorter interconnects. With such interconnects, scale-in clusters consume less energy for remote data accesses compared to their scale-out cluster counter-parts. To shed some light on the benefits, we evaluate the energy efficiency gain of scale-in clusters.

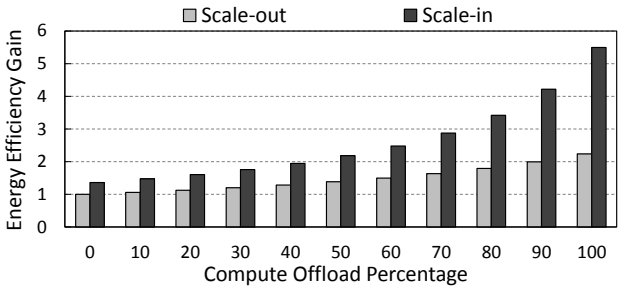

Figure 7: Energy consumptions and efficiencies of an existing scale-out system and a lower-degree scale-out system that consists of fewer number of nodes with ISP. This proposed system can achieve 2X energy efficiency with 50% execution of big data analytics at ISP devices. With 80% compute offload, the relative energy efficiency as high as 3.4X compared to the scale-out clusters.

Compared to conventional scale-out clusters, scale-in ISP clusters can significantly improve the efficiency with fewer nodes. By varying the ratio of compute offload to ISP devices, Figure 7 shows relative energy efficiency gains compared to the energy efficiency of scale-out clusters without ISP, based on 7nm technology projections. Left bars and right bars are distinguished by the size of clusters, i.e. degrees of scaling-out: fewer nodes are populated with ISP in the scale-in case. Here we assume the same job in Section 4.2, but with 10 iterations of the computation. We also assume 25X higher energy efficiency of ISP computation as described earlier. We assume that I/O reduction rate is proportional to the compute offload. Note that we assume the average distance for remote data accesses is reduced down to 50m from 100m due to the shrunken physical dimension of a system with scaling-in.

Scale-out clusters with ISP shows efficiency gains up to 2.2X compared to the efficiency of clusters with conventional servers by utilizing more efficient computation at ISP devices with reduced local data movements. That is, ISP's task executions reduce the energy consumption for computation and suppresses the energy consumption of local data read/write. Moreover, the energy efficiency gain further increases up to 5.5X with scale-in clusters by reducing remote data movements as well. Shorter interconnects in this system contribute up to 59% of the efficiency gain by reducing the energy cost of remote data accesses (comparing scale-out and scale-in clusters with 100% compute offload). In overall, scale-in clusters with ISP achieve 2.2X energy efficiency with 50% compute offload and 3.4X energy efficiency with 80% compute offload. Ideally, the overall energy efficiency can be improved up to 5.5X with 100% compute offload to ISPs in data-intensive applications.

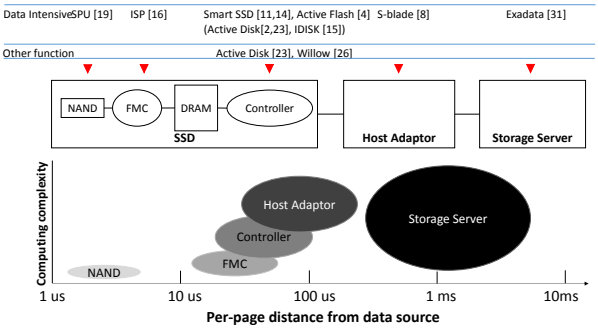

Figure 8: Data latency, system component, and computing complexity for ISP.

6. SURVEY OF IN-STORAGE PROCESSING

The idea of using storage device as a supplementary processing unit was first introduced a few decades ago for database machines. Early systems mainly focused on improving the performance of hard disks by dedicating a processor per head, track, or disk [22, 28], but marginal performance improvement could not justify the high manufacturing cost.

In the late 1990s, this idea was resurrected as computer systems were commoditized. Riedel et al. proposed Active Disk [23] that combines on-drive processing and large memory to allow disks to execute application-level functions in the device. In the meantime, Acharya et al. focused on the programming model and algorithms rather than the device architecture [2]. Afterward, Keeton et al. introduced IDISK [15] designed as a general purpose network node that can replace costly cluster nodes for scalable decision support databases. However, none of these were implemented on commercially available hardware because of the physical space constraint, the limited power and cooling supply issues.

Later, near-data processing have been realized with special-purpose or commodity hardware to improve the performance of database processing. Mueller et al. [20] proposed a FPGA-based approach. Data being pre-processed, the amount of data delivered to the hosts is reduced and the host can leverage low memory footprint and high cache-hit rate. Commercial products based on this idea found in IBM Netezza's S-Blade server [8] and Oracle's Exadata [31].

Recently, the advances in SoC (System on Chip) and SSD (Solid State Drive) technologies have the resurgence of interest in ISP. Kim et al. [16] investigated the benefits on SSD with the database scan operation. Since the fast storage medium and the increased parallelism in SSDs make the embedded processors and the memory in SSD the bottleneck, the scan operation is implemented inside a FMC (Flash-Memory Controller) that is responsible for driving I/O commands to the NAND packages on a channel as shown in Figure 8. Due to the limited processing power, the limited amount memory, and real-time constraints in a FMC, its applications were limited to relatively simple data processing such as data filtering.

In the meantime, Boboila et al. proposed Active Flash [4] to enable out-of-core data analytics, providing an analytical model that shows the performance-energy tradeoffs in moving data processing to SSDs. Peng et al. [19] from Seagate Technology proposed a concept of storage processing unit (SPU). SPU basically adds a coprocessor in the pad-limited die inside the NAND flash package. This would be one extreme solution that moves the compute closest to the data and a simulation-based experiments show more than 100 times energy efficiency.

Kang et al. [14] and Do et al. [11] presented the first end-to-end experiments of analytic processing in big data (*i.e.,* Hadoop) and relational queries (*i.e.,* MS SQL server) by using commercial commodity hardware. The results proved that significant performance and energy gains can be achieved by pushing selected query processing or analytic processing components inside a SSD.

Latestly, Seshandri et al. proposed Willow [26]. Contrast to the prior works that mostly focused on data intensive computing in the pursuit of near-data processing, Willow focused on different programmability and flexibility of SSDs such as data-dependent read/write to reduce outstanding messages between host and device, semantic extensions for caching or logging, and privileged execution taking over operating system or file system functions.

In summary, there has been a wide spectrum of studies realizing ISP from a chip-level approach [19] to pure host-based approach [31] as shown in Figure 8. Each approach has different budgets on latency and power due to the distance from the data source, which limits the complexity that it can handle without compromising the performance. For instance, too many gates in pad-limited die can increase timing of NAND chips which impact latency, overall throughput, and power consumption. Likewise, embedding high-performance CPUs into SSDs is also challenging to meet a power budget (e.g. 15W for SAS and 25W for PCIe).

ISP using SSDs has several advantages over the old HDD-based ones. It can exploit the SSD internal components and improve the I/O performance by leveraging internal parallelism efficiently: multiple independent I/O channels and embedded processors. Unlike intelligent disks, the perfor-

mance of ISP is not as affected by data fragmentation that deteriorates the I/O performance, and powerful processors and large memory capacity can handle more complex operations such as join and merge. When such devices are properly coordinated with host via a well-defined mechanism as in HSA (Heterogeneous System Architecture) [25] or an AMPP (Asymmetric Massive Parallel Processing) [8], ISP can be a viable building block to architect a balanced system with a better TCO (Total Cost of Ownership) model.

7. CONCLUSION AND FUTURE WORK

Big data analytics benefit from scale-out clusters along with data-intensive programming and execution frameworks. However, unlike traditional computing environment, file I/O becomes a significant bottleneck toward energy efficient computing. Comparing the energy consumption of data movements to its computation energy, we show that scale-in clusters with ISP would achieve much higher energy efficiency, potentially quadrupling the efficiency of the current scale-out clusters.

To achieve such energy-efficient ISP architectures, we need to investigate several areas further. ISP designs, host interfaces, and programming models should be considered and related technical issues need to be solved over times. For example, although previous ISP studies explored some of these challenges, it is still in its early stage and yet to be matured. Thus, real-world issues, such as data security, task scheduling, compilers, and run-time environments should be also investigated.

8. REFERENCES

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *ACM SIGOPS Operating Systems Review*, volume 32, pages 81–91. ACM, 1998.

[2] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VIII, pages 81–91, New York, NY, USA, 1998. ACM.

[3] G. Ananthanarayanan and I. Menache. Big data over networks.

[4] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman. Active flash: Out-of-core data analytics on flash storage. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–12. IEEE, 2012.

[5] S. Borkar. Role of interconnects in the future of computing. *Journal of Lightwave Technology*, 31(24):3927–3933, 2013.

[6] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[7] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger. Active disk meets flash: a case for intelligent ssds. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 91–102. ACM, 2013.

[8] S. Cputo, J. Tuckwell, and L. Patterson. Ibm puredata system for analytics architecture: A platform for high performance data warehousing and analytics. *IBM Corporation*, 2010.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] R. Dementiev, T. Willhalm, O. Bruggeman, P. Fay, P. Ungerer, A. Ott, P. Lu, J. Harris, P. Kerly, and P. Konsor. Intel performance counter monitor, 2012.

[11] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt. Query processing on smart ssds: opportunities and challenges. In *Proceedings of the 2013 international conference on Management of data*, pages 1221–1230. ACM, 2013.

[12] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 37–47, New York, NY, USA, 2010. ACM.

[13] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[14] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park. Enabling cost-effective data processing with smart ssd. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pages 1–12. IEEE, 2013.

[15] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A case for intelligent disks (idisks). *SIGMOD Rec.*, 27(3):42–52, Sept. 1998.

[16] S. Kim, H. Oh, C. Park, S. Cho, and S.-W. Lee. Fast, energy efficient scan inside flash memory ssds. In *Proceeedings of the International Workshop on Accelerating Data Management Systems (ADMS)*, 2011.

[17] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[18] H. Li, A. Ghodsi, M. Zaharia, E. Baldeschwieler, S. Shenker, and I. Stoica. Tachyon: Memory throughput i/o for cluster computing frameworks. *memory*, 18:1, 2013.

[19] P. Li, K. Gomez, and D. J. Lilja. Exploiting free silicon for energy-efficient computing directly in nand flash-based solid-state storage systems. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[20] R. Mueller, J. Teubner, and G. Alonso. Data processing on fpgas. *Proc. VLDB Endow.*, 2(1):910–921, Aug. 2009.

[21] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI) Oakland, CA*, pages 293–307, 2015.

[22] E. A. Ozkarahan, S. A. Schuster, and K. C. Smith.

Rap - an associative processor for database management. In *AFIPS National Computer Conference'75*, pages 379–387, 1975.

[23] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 62–73, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[24] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 371–382. ACM, 2009.

[25] P. Rogers and A. C. FELLOW. Heterogeneous system architecture overview. In *Hot Chips*, 2013.

[26] S. Seshadri, M. Gahagan, S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson. Willow: A user–programmable ssd. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 67–80. USENIX Association, 2014.

[27] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 59(1):7–1, 2015.

[28] S. Y. W. Su and G. J. Lipovski. Cassm: A cellular system for very large data bases. In *Proceedings of the 1st International Conference on Very Large Data Bases*, VLDB '75, pages 456–472, New York, NY, USA, 1975. ACM.

[29] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 231–242. ACM, 2010.

[30] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 488–499. IEEE, 2014.

[31] R. Weiss. A technical overview of the oracle exadata database machine and exadata storage server. *Oracle White Paper. Oracle Corporation, Redwood Shores*, 2012.

[32] L. Wilson. International technology roadmap for semiconductors (itrs). *Semiconductor Industry Association*, 2013.

[33] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.