In-Storage Computing for Hadoop MapReduce Framework

Dongchul Park, *Member, IEEE,* Kwanghyun Park, *Member, IEEE,* Yang-Suk Kee, *Member, IEEE,* and Jignesh M. Patel, *Fellow, IEEE*

Abstract—Solid State Drives (SSDs) have been developed for the replacement of conventional magnetic Hard Disk Drives (HDDs) as a faster storage device. Their high computational capabilities, however, enable SSDs to become a computing node, not just another faster storage device, which is called In-Storage Computing (ISC). Hadoop MapReduce framework nowadays became a de facto standard for big data processing. This paper explores the challenges and opportunities of In-Storage Computing for Hadoop MapReduce framework. For this, we co-design a Hadoop MapReduce system and an ISC device by implementing the Mapper inside real SSD firmware and offloading Map tasks from a host MapReduce system to the SSDs. Then, we make extensive experiments in real Hadoop clusters. Our experiment results demonstrate that our ISC Hadoop MapReduce system achieves a remarkable performance gain (2× faster) as well as significant energy saving (9.3× lower).

Index Terms—In-Storage Computing, Storage Intelligence, In-Situ Processing, Near data processing, Smart SSD, Hadoop, MapReduce, Big Data.

1 Introduction

Due to the superior characteristics of Solid State Drives (SSDs) to the rotational Hard Disk Drives (HDDs), SSDs have been increasingly adopted even in enterprise systems as well as in personal/mobile systems [1], [2], [3]. Originally, SSDs have been developed for the purpose of replacing the traditional HDDs as a faster storage device since, for instance, modern SSDs are over 100 times faster than HDDs in accessing data. Moreover, they consume over 10 times less energy than HDDs [4]. Thus, most of the recent research studies have focused primarily on how to make use of SSDs as yet-another-faster HDDs.

However, high computational capabilities of the modern SSDs had people start to rethink of SSDs as, not just a faster storage device, but another type of a computing node, so-called In-Storage Computing (for short, ISC). Unlike a traditional CPU-centric computing philosophy–"moving data closer to codes", this ISC model suggests a totally different computing paradigm–"moving codes closer to data" thereby offloading key functions from a host system to a device, which significantly changes a data flow for computation [5]. This data flow change enables the ISC system to achieve not only a faster performance but also a marvelous energy saving, which results in the notable savings of Total Cost of Ownership (TCO) in the long run. This ISC computing paradigm is newly spotlighted in the big data era these days.

- *Dongchul Park and Yang-Suk Kee are with the System Architecture Lab. (SAL) of Samsung Semiconductor Inc., San Jose, CA 95134. E-mail: {dongchul.p1, yangseok.ki}@ssi.samsung.com*
- *Kwanghyun Park and Jignesh M. Patel are with the Department of Computer Science, University of Wisconsin–Madison, Madision, WI 85123. E-mail: kpark, j.patel@cs.wisc.edu*

Manuscript received April 19, 2005; revised September 17, 2014.

Hadoop MapReduce is a software framework for the distributed processing of large data sets on clusters of commodity hardware, and MapReduce framework nowadays became a de facto standard for big data processing [6]. In the MapReduce framework, computation is divided into two functions: Map and Reduce. Mapper takes a set of data and converts it into another set of data (i.e., intermediate data), where individual elements are composed of key/value pairs. The Reducer takes the outputs from Mappers as input and combines those intermediate data into a smaller set of key/value tuples [7]. The Hadoop MapReduce system tries to collocate the data in a local computing node to take advantage of data locality, which is at the heart of MapReduce and is the reason for its good performance [8]. This Hadoop policy–"putting computation near the data" aligns well with the aforementioned recent computing paradigm shift, which was also early advocated by Jim Gray [9].

This work explores the challenges and opportunities of In-Storage Computing for the Hadoop MapReduce framework. We offload Mapper into the ISC device (i.e., SSD) by implementing Map features inside real SSD firmware and integrate our ISC Hadoop framework into the existing Hadoop MapReduce system framework. We set up Hadoop clusters and run a Hadoop MapReduce application on the clusters to verify our holistic ISC Hadoop framework. Our experiment results show that the ISC Hadoop system achieves a remarkable performance gain (2× faster) and significantly reduces energy consumption (more than 9× lower).

The system co-design of our ISC Hadoop MapReduce causes several interesting and very challenging issues as follows.

Discrepancy in data representation: For data access, Hadoop in the host uses file systems such as Hadoop Distributed File System (HDFS) and a local file system such

as EXT3/4. However, an ISC device cannot rely on any file system information and, instead, uses different data representation such as a Logical Block Address (LBA). Therefore, a host Hadoop system should be able to collaborate with ISC devices by employing LBA information, not using any file systems.

Discrepancy in system interfaces: System interfaces between the Hadoop framework in the host system and an ISC device can be different. That is, a host Hadoop framework uses Java programming language, while our ISC framework inside an SSD device can adopt a different language like C/C++. As a result, the host Hadoop system cannot directly communicate with ISC devices.

Data split: Hadoop Distributed File System (HDFS) splits large data into a unit of 64MB (i.e., input split) by default [8]. This file split process gives rise to a serious data split issue in the ISC model. As an example, HDFS may split the word 'Hadoop' into 'Ha' and 'doop' during the aforementioned input split process. In fact, this does not causes any issue in a typical Hadoop system because HDFS in the host Hadoop system loads large data in the main memory and processes them in a streaming data access manner. On the other hand, our ISC Hadoop MapReduce framework basically does not move data from devices to the host memory. Thus, it inherently cannot avoid this data split issue in the Hadoop MapReduce framework.

Feature offloading: We move only Mapper, not both, from a host to an ISC device. Unlike a Map task that does not have a dependency among other tasks, a Reduce task collects the output results of other Map tasks in its shuffle phase. However, since our ISC device cannot directly communicate with other ISC devices, a host system should be in charge of collecting the intermediate data from each Mapper and, moreover, redistribute the collected data to each ISC device for the Reduce execution. This incurs unnecessary redundant data movement and the ISC devices can be overwhelmed due to the limited resources.

A fully distributed mode on a single node: Hadoop basically does not support a fully distributed mode on a single node (i.e., multiple instances of datanode on a single machine). Only one instance of datanode can be configured in a pseudo distributed mode. However, it can provide a very useful and efficient way by simulating the fully distributed Hadoop clusters with only a single machine. We did a workaround for this mode and all our initial studies are done with this mode.

We address all these challenges in the design section (Section 3) in more detail. The main contributions of this work are as follows:

- ***Real SSD implementation***: We implemented ISC features (i.e., Mapper) inside real SSD firmware. All experiments represent a real device and system performance.
- ***Hadoop system integration***: We integrated our ISC Hadoop framework with the existing Hadoop MapReduce framework to seamlessly support the existing Hadoop features.
- ***Exploration of challenging issues***: The ISC model for Hadoop MapReduce framework causes the aforementioned challenging issues. We judiciously tackled those issues in our proposed ISC Hadoop framework.

The rest of this paper is organized as follows. Section 2 provides an overview of In-Storage Computing and its architecture as well as a Hadoop MapReduce framework. Section 3 presents our proposed ISC Hadoop MapReduce framework, and explains our system design and challenges. Section 4 shows our extensive experiment results and provides analyses to demonstrate the performance gain of our ISC Hadoop system. Section 5 discusses related studies of this work, and Section 6 concludes our work.

2 Background

This section describes the background of SSD-based In-Storage Computing (Section 2.1) and a Hadoop MapReduce framework (Section 2.2).

2.1 In-Storage Computing (ISC)

In-Storage Computing is a new computing paradigm, where a storage device play a major role in data processing as a computing node. Unlike the traditional CPU-centric computing model, nowadays non-CPU components such as storage devices or Graphical Processing Unit (GPU) can make a big contribution to data computation by using their computing capabilities [10], [11], [12], so-called *'rebellion of peripherals against CPU'*.

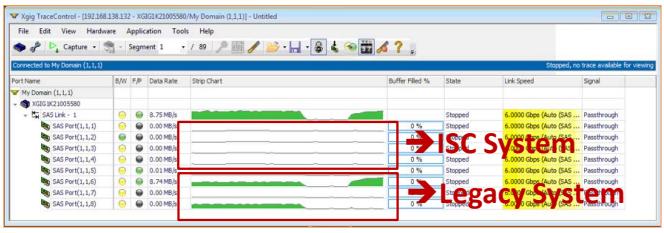


Fig. 1: The amount of data transfer from a device to a host. The upper most green bar corresponds to the total amount of data for both systems.

Figure 1 shows the amount of data transferring from a device (SSD) to a host system and we captured this with our 12Gbps SAS/SATA bus analyzer [13]. For this study, we set up two Hadoop systems (i.e., our proposed ISC Hadoop system and typical Hadoop system with SSDs) and connect them to the bus analyzer. Then, we run the same Hadoop application with identical data on both Hadoop systems at the same time. As shown in the figure, both exhibit a huge difference between our ISC model (upper rectangle) and the traditional CPU-centric computing model (lower rectangle). This clearly presents that the ISC system does not (or very rarely) transfer data to the host system for computation, while the legacy CPU-centric system keeps loading all the data to the host system (that is, host DRAM). All the main benefits of the ISC system originate from this factor.

Even though the main idea of ISC has been already proposed and implemented in the context of Hard Disk Drives (HDDs), it was not successfully adopted because of the still very limited computational capabilities of HDDs [14], [15]. However, the modern SSDs have been equipped with even stronger computing resources such as multi-core CPUs and DRAM so that people start to rethink of SSDs as another type of a computing unit, not just as a faster HDDs.

SSDs typically provides a lot higher internal bandwidth (about 5× higher) than host I/O interface (SATA or SAS) bandwidth [12], [16]. The SSDs adopted for our ISC Hadoop system offers about 3.2GB/s of the aggregated internal bandwidth, while 750MB/s of host I/O interface bandwidth[1]. Thus, moving data from devices to a host results in a significant waste of the high internal bandwidth. In addition, the low I/O latency inside SSDs is another noticeable factor. A regular I/O operation is affected heavily by the entire OS stack which introduces extra overheads (e.g., context switching, interrupt, file system overhead, etc). However, the internal I/O latency in SSDs can avoid those OS software overheads. These factors implies that *I/O intensive applications can significantly benefit from the ISC model* because, unlike CPU-centric computing system, ISC systems can make the best use of the high internal bandwidth and low internal I/O latency.

SSDs, in general, are equipped with low power processors not only to save energy consumption but also to lower manufacturing costs. This factor results in an ambivalent value of ISC computing model. That is, the low power processor such as ARM processors can help ISC applications save energy consumption. On the other hand, it implies *CPU computation intensive applications are not favorable to our current ISC computing model*.

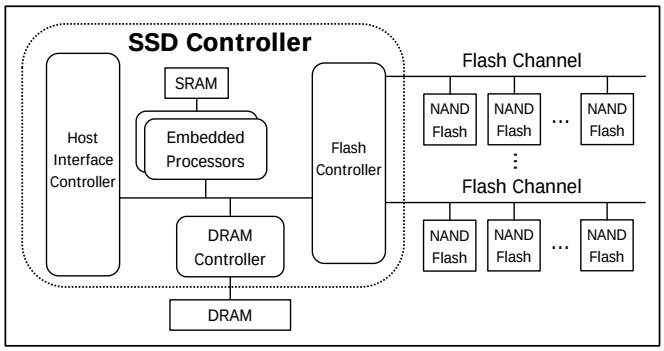


Fig. 2: ISC hardware architecture

2.1.1 ISC Hardware Architecture

Figure 2 represents the hardware architecture of the modern SSD. The hardware architecture of our ISC SSD device is identical to regular SSDs.

An SSD is largely composed of NAND flash memory array, SSD controller, and (device) DRAM. The SSD controller is subdivided into four main subcomponents: host interface controller, embedded processors, DRAM controller, and flash controller. The host interface controller processes commands from interfaces (typically SAS/SATA or PCIe) and distributes them to the embedded processors. Commands come from a user through the host I/O interface and the common interfaces are implemented by the host interface controller. The embedded processors receive the commands and pass them to the flash controller. More importantly, they run SSD firmware codes for computation and execute Flash Translation Layer (FTL) for logical-to-physical address mapping [17], [18]. Typically, the processor is a low-powered 32-bit processor such as an ARM processor. Each processor can have a tightly coupled memory (i.e., SRAM) to store performance-critical data or codes. Each processor can access DRAM through the DRAM controller. The flash controller controls data transfer between flash memory and DRAM.

The NAND flash memory package is the persistent storage media and each package is subdivided further into smaller units that can independently execute commands or report status. An SSD is also equipped with a large size of DRAM for buffering data or storing metadata of the address mapping. All the flash channels share access to the DRAM. Thus, data transfer from the flash channels to the DRAM needs to be serialized [12].

1. Both are theoretical bandwidths and based on Samsung 6Gb SAS Enterprise SSD (400GB SLC).

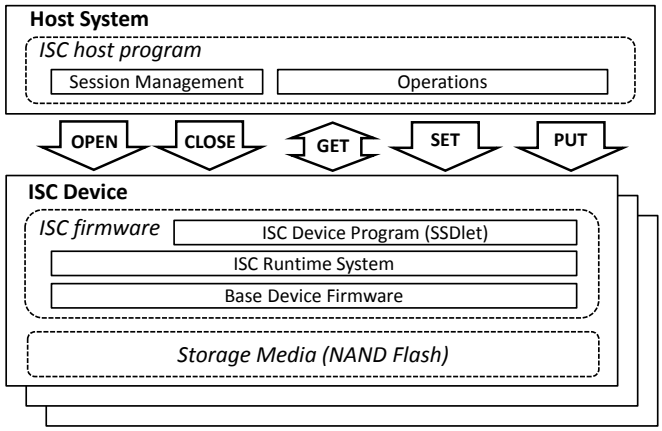


Fig. 3: ISC software architecture

2.1.2 ISC Software Architecture

This section describes the system architecture and software key components for our ISC ecosystem including host system and storage device. Figure 3 represents our ISC software architecture composed of two main components: *ISC firmware* inside the SSD and *ISC host program* in the host system. The ISC host program communicates with the ISC firmware through ISC Application Programming Interfaces (APIs).

Since both a host system and an ISC storage device cooperatively compute data, we need to define two types of interactions (i.e., ISC host interface and ISC device interface) among a host, a device, and an ISC application. The ISC host interface is in charge of communication between the host and ISC devices to control ISP operations. The ISC device interface is on how the ISC runtime system in the storage device internally interacts with an ISC application in response to external inputs. A host ISC program implements the host interface logic, while a device ISC program (i.e., SSDlet) implements the device interface logic. We implement our ISC applications by using the ISC APIs.

The ISC firmware is subdivided into three subcomponents: SSDlet, ISC runtime, and base device firmware. An SSDlet is an ISC device program inside the SSD. It implements application logic and responds to the ISC host program. The SSDlet is executed in an event-driven manner by the ISC runtime system. An ISC runtime system connects the ISC device program with a base device firmware, and

implements the library of ISC APIs. In addition, a base device firmware also implements normal I/O operations (read and write) of a storage device.

After the SSDlet is installed in the ISC device, a host system runs the ISC host program to interact with the SSDlet in the devices. This ISC host program consists largely of two components: a session management component and an operation component. The session component manages the lifetime of a session for ISC device applications so that the ISC host program can launch an SSDlet by opening a session to the ISC device. To support this session management, our ISC programming model provides two APIs, namely, OPEN and CLOSE. OPEN starts a session and CLOSE terminates a session. Once OPEN starts a session, the runtime resources such as memory and threads are assigned to run the SSDlet and a unique session ID is returned to the ISC host program. When CLOSE terminates the established session, it releases all the assigned resources and closes SSDlet associated with the session ID.

Once a session is established by OPEN, the operation component helps the ISC host program interact with SSDlet in an ISC device with GET, SET and PUT APIs. This GET operation is used to check the status of SSDlet and receive output results from the SSDlet if the results are ready. This GET API implements the polling mechanism of the SAS/SATA interface because, unlike PCIe, such traditional block devices cannot initiate a request to a host such as interrupts.

Aside from the existing OPEN, GET, and CLOSE APIs, *we implemented two more APIs: SET and PUT*. The SET operation delivers the new ISC metadata information (e.g., starting LBA, data length, etc.) from a host to an ISC device within a session. This SET API is very useful to process multiple non-contiguous data within one session. Without the SET API, we had to re-open another session after closing the previous session, which causes an extra overhead. PUT is also newly implemented to internally write data to the NAND flash media without help of a local file system. This API is useful to store the intermediate data during ISC processing.

2.2 Hadoop MapReduce Framework

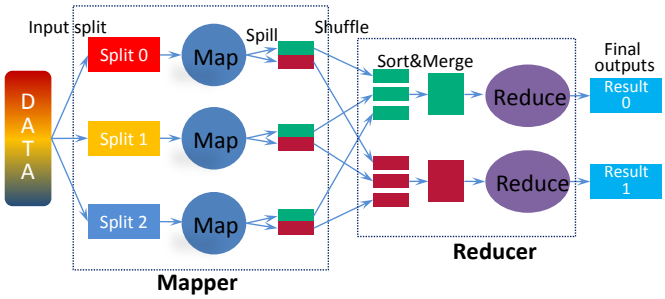


Fig. 4: Hadoop MapReduce Framework

Hadoop is an open source implementation of the original MapReduce parallel programming model and system designed by Google [7]. MapReduce is a software framework for distributed processing of large data sets on commodity clusters [19]. The Hadoop MapReduce framework leverages a distributed file system called the Hadoop Distributed File System (HDFS) which is an open source implementation of Google Distributed File System (GFS) [20]. The main design of the HDFS is heavily optimized for manipulating very large files with streaming data access patterns (i.e., write-once, read-many patterns). A MapReduce job consists of both Map and Reduce tasks, and one complete MapReduce job is subdivided largely into three phases: Map, Shuffle, and Reduce phase.

As illustrated in Figure 4, at the beginning of a Hadoop MapReduce processing, Hadoop splits large input data into a set of data chunk of a predefined size (by default 64MB). Each split data is fed to each Map task which is executed in parallel on many physical machines. The Map tasks apply the Map function to its own input split data and generate a set of output data (i.e., intermediate key-value pairs). If a user implemented an optional combine function, Map tasks apply this combine function to each key-value pair list at this moment. Now, each Map task partitions the intermediate data (key-value pairs) into the number of Reduce groups. We offload this Map function into the ISC device.

Once all Map tasks are successfully completed, Hadoop MapReduce framework transfers the Map outputs to the Reducers as inputs, which is known as the Shuffle. Each Reduce task requests one intermediate data file from each Map task and starts to merge them on the basis of the intermediate key. Someone think of this Shuffle phase simply as a part of Reduce phase and they call it a 'copy' step in the Reduce phase.

After all Reduce tasks copy their own intermediate data, they start to sort and merge them first. Then they apply a Reduce function to the sorted data and emit the final output for each Reducer. Reduce tasks write their final outputs to a file with Hadoop Distributed File System (HDFS).

3 ISC Hadoop MapReduce

This section describes the system co-design of our In-Storage Computing model and Hadoop MapReduce framework.

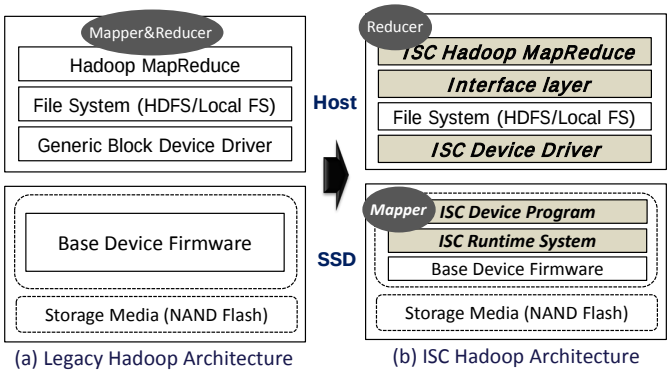


Fig. 5: ISC Hadoop MapReduce architecture

3.1 System Architecture

Figure 5 shows an overall system architecture of our ISC Hadoop MapReduce framework. As explained, we offload the Mapper into an ISC device and implement the Mapper feature inside the real SSD firmware. Thus, a host Hadoop

MapReduce system must be revised to collaborate with our ISC device thereby disabling Mapper features in the host system and implementing them inside SSD firmware (i.e., ISC device program). To support ISC functions, we also need to develop ISC engines (i.e., ISC Runtime System) on top of the existing base device firmware. The interface layer plays a role of an agent in communication between the host Hadoop MapReduce system and our ISC MapReduce system. We suggest this software component to resolve two major design issues: discrepancy in system interfaces and discrepancy in data representation between the host and the ISC devices (these issues will be addressed in the subsection 3.3.1 and 3.3.2 in detail.). Alongside of a generic SCSI block device driver, we implement a special SCSI command to trigger ISC features inside the SSD (ISC Device Driver). We do not touch both HDFS and a local file system. So, our ISC Hadoop MapReduce system seamless supports all existing Hadoop features.

We first explore our work on a single Hadoop machine to verify our idea and concept, and then extend our design to Hadoop clusters of multiple nodes.

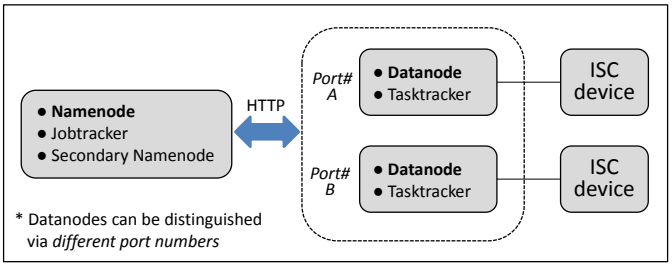


Fig. 6: ISC Hadoop configuration on a single node: a fully distributed mode on a single machine

3.1.1 A Single Node

There are typically three ways to install and configure Hadoop: a local (standalone) mode, a pseudo distributed mode, and a fully distributed mode. In the standalone mode, all Hadoop components (i.e., namenode, jobtracker, datanode, tasktracker, secondary namenode) run in a single Java process (i.e., JVM instance) and this mode does not use HDFS. In the pseudo distributed mode, Hadoop runs all the components as separate Java processes and uses HDFS. This serves as a simulation for fully distributed Hadoop clusters but it runs *a single instance of datanode on a single machine* only. The fully distributed mode runs on clusters of multiple machines with a master (plays namenode and job tracker. Optionally secondary namenode) and slaves (plays datanodes and tasktrackers) concept [6].

As described, basically it is impossible to run Hadoop in a fully distributed mode on a local machine (i.e., *multiple instances of datanode on a single machine*). We, however, install and configure Hadoop in the fully distributed mode on a single node with a workaround, which is very efficient to initially verify our proposed ISC Hadoop MapReduce framework. Figure 6 illustrates this Hadoop configuration on a single node. We configure separate Hadoop configuration files for each datanode instance. For HTTP communication with the namenode, we assign different port numbers for each datanode instance, wherein each datanode connects with our ISC device. Thus, each datanode can be distinguished via different port numbers for communication with the datanode[2]. This workaround mode represents the fully distributed Hadoop clusters on a single node.

3.1.2 Clusters: Multiple Nodes

The aforementioned single node configuration represents well the main features of the distributed Hadoop clusters on a local machine. Based on this observation, we extend our ISC Hadoop MapReduce framework to real Hadoop clusters. We set up the clusters of 5 nodes and separate a namenode from other datanodes (i.e., 1 namenode and 4 datanodes) to completely eliminate its performance overheads from the datanode. We initially start our study with one datanode and gradually increase the number of datanode to four on our clusters. Since we found meaningful and predictable performance results (that is, almost linear performance increase), we did not set up the bigger Hadoop clusters (please refer to our experiment results).

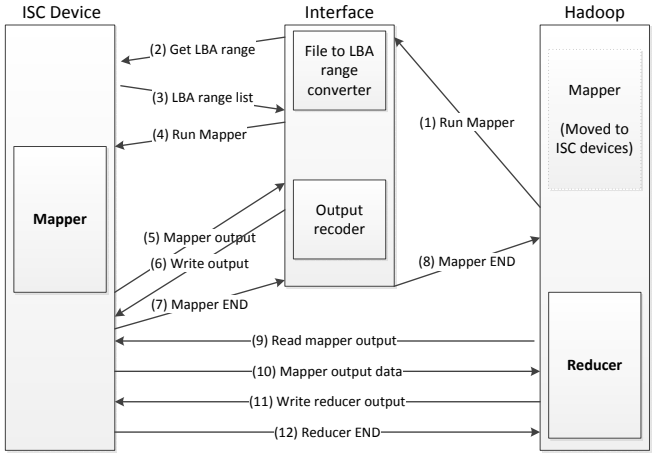


Fig. 7: ISC Hadoop MapReduce workflow

3.2 Workflow

Figure 7 depicts our ISC Hadoop MapReduce workflow. Once a Hadoop application runs, our ISC Hadoop MapReduce framework does not directly run Mappers in the host system because they moved inside ISC devices. Instead, it first calls an interface component to communicate with ISC devices while delivering the input split data information (i.e., a file name of the input split) for ISC Map tasks. The interface runs its software component for converting each input split data (64MB) information to different data representation (i.e., LBA range information) for ISC devices. We found each input data of 64MB is generally split further into 2 or 3 smaller data chunks via an EXT4 file system. This means the software component in the interface (i.e., File to LBA range converter) delivers 2 or 3 LBA range lists to an ISC device for each Map task. That is, the interface component calls Mapper inside the ISC device with these converted LBA range lists. After the Mapper receives all metadata information from the interface layer, it starts to

2. This configuration is based on Hadoop 0.20.205 and we found the assignment of virtual IPs to each datanode instance did not work for this workaround in this Hadoop version.

execute a Map task. Once the Map task completes, the ISC devices return their Mapper outputs (i.e., intermediate results) to the interface layer to save them in the device (that is, the Mapper outputs are written with a local file system.). In fact, these Map output writing processes (Step 5 and 6) can be replaced with our new ISC API–PUT. Now, Reducers in the host can start to read the intermediate data as their input values and perform Reduce tasks. Finally all Map and Reduce tasks completes, our Hadoop system stores the final results with Hadoop Distributed File System (HDFS).

3.3 Design Challenges

This section addresses the challenging issues to design our ISC Hadoop MapReduce framework.

3.3.1 Discrepancy in Data Representation

To access data in a device, a host system can use file system information and so does the host Hadoop system by employing HDFS or local file systems such as EXT3/4. However, the device itself cannot rely on any file system information to access data inside its own device for ISC processing. This discrepancy in data representation between the host and the device causes an essential problem. This is the main reason the previous ISC work [12], [21] had no choice but to write their data to the raw device with 'dd' command by explicitly specifying their data locations (LBAs) from the beginning. This approach can temporarily evade this issue; but cannot resolve it. Consequently, it gives rise to other critical limitations on their systems. For instance, Kang et al. proposed a Hadoop MapReduce system leveraging ISC computation [21]. However, due to this challenging issue, their Hadoop system works only one local machine (i.e., no scalability) since it cannot support HDFS and local file systems.

We fundamentally figure out this critical issue by developing a software component in the interface layer between the Hadoop system and ISC devices (please refer to the Figure 7). This component is in charge of converting a file name into LBA range list information for the ISC device. Therefore, our proposed ISC Hadoop MapReduce system can seamlessly support the existing Hadoop features. For this, we first get a local file system information of each data split (64MB) from HDFS and convert it to LBA range lists. Then we deliver this information to ISC devices for Map tasks.

3.3.2 Discrepancy in System Interfaces

System interfaces between Hadoop framework in the host system and ISC device can be different. That is to say, Hadoop MapReduce framework in the host adopts Java programming language, while our SSD firmware for ISC processing uses C/C++ language. As a result, the host Hadoop system cannot directly communicate with ISC devices. To resolve this issue, we can think of two options: (1) running OS inside ISC devices and (2) adopting an interface layer. If an OS runs inside ISC device, it can also simply run JVM on top of the OS. This solution looks like very intuitive and may be able to eliminate our aforementioned challenge (i.e., discrepancy in data representation). However, this approach requires to consume extra ISC computing resources such as CPU and DRAM inside the device. Thus, unless the ISC device is a very specialized device with powerful computing resources, the first option will not be a good choice.

As a matter of fact, since our ISC device is a typical commodity SSD without enough computing resources, we choose the second option. We, instead, put another interface layer between the Hadoop system and ISC devices by adopting Java Native Interface (JNI). Aside from the programming system interface between them, this interface layer plays two important roles in our ISC Hadoop System–first, as mentioned above, converting a file name to LBA range information, and second, writing the output data (i.e., intermediate data) of ISC Map tasks. For this, the host Hadoop system needs to deliver metadata information including an input split file name and an output file name of the intermediate data. The overhead of this extra interface layer is almost ignorable.

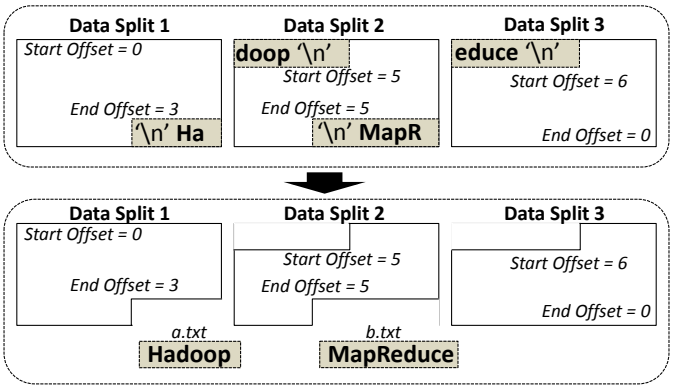


Fig. 8: A data split example

3.3.3 Data Split

Hadoop Distributed File System (HDFS) splits large input data into a chunk of 64MB by default [8]. This input split process raises another critical problem in ISC Hadoop system: data split. Figure 8 illustrates this issue. Assuming a word data 'Hadoop' is stored in the large input data, HDFS may split the word 'Hadoop' into 'Ha' in one file of 64MB (i.e., Data Split 1) and 'doop' in another file of 64MB (i.e., Data Split 2) if the word inadvertently lies in the input split boundary. This does not causes any issue in a typical Hadoop system because HDFS in the host Hadoop system loads all input split data in the main memory and processes them in a streaming data access manner. However, since ISC Hadoop MapReduce framework basically does not move input data from devices to the host memory, this data split issue can cause a critical problem in the ISC Hadoop system. If the ISC device is equipped with such a capability that they can directly connect/communicate with each other like Ethernet-connected drives (e.g., Seagate's Kinetic HDD or HGST's Ethernet Drive [22], [23]), we may be able to solve this issue with a naive way, but unfortunately, our ISC drive is not equipped with such a communication interface.

To resolve this, our ISC Hadoop system consults the host Hadoop system. If the host Hadoop system detects a split data[3], it stores the split data (both Hadoop and

3. Host Hadoop system originally has a simple algorithm for this data split detection.

MapReduce in the example) as a separate file respectively with a local file system. Then it converts the file name to LBA information by using the aforementioned software component and delivers the LBA information to the ISC device via the interface layer. To avoid inaccurate (i.e., duplicated) computation of that part inside ISC device, the host system provides the ISC device with another metadata information (that is, Start Offset and End Offset) for each data split. These offsets can be thought of as data lengths of each part including a delimiter, where the delimiter can be a carriage return, space, or tab, depending on the Hadoop applications. With the help of this metadata information, ISC Mapper can simply ignore processing all the data within the offset boundary in each data split chunk of 64MB.

3.3.4 Feature Offloading

The data precessing of Hadoop MapReduce framework is composed of Map and Reduce. Unlike a Map task that does not have any dependency among other tasks for its execution, a Reduce task is inherently dependent on other Map task executions. That is, a Reducer is required to collect output results of other Map tasks in its shuffle phase before a Reduce task starts. However, as we mentioned in the previous subsection 3.3.3 (Data Split), our ISC device is not equipped with such a communication capability, a host system should be in charge of collecting the intermediate data from each Mapper. Moreover, the host should redistribute the collected data to each ISC device for the Reduce execution. Not only can these processes incur unnecessary data movement traffic, but also can the ISC devices be overwhelmed by the overburden because typically an ISC device has very limited resources and both Mapper and Reducer, in general, are executed concurrently.

Many Hadoop MapReduce applications (not necessarily, but mostly) produce small amount of intermediate data after a Map task processes original big data. In addition, the Reduce task is subdivided into three major phases such as copy, sort, and reduce. Unlike a Map task, the Reduce task generally does not significantly lessen the data size of its output result after Reduce processing. This is not favorable to ISC framework [12]. The sort can give rise to another unfavorable issue. Assuming a data size is bigger than an available DRAM space in an ISC device, it may have to issue tremendous NAND flash Write operations to store intermediate results, which has a critical impact on the life endurance of NAND flash memory [24], [25], [26], [27].

Based on these observations, we move Mapper to ISC devices, not Reducer, in the Hadoop MapReduce framework.

3.4 Hadoop Applications

As discussed in the Section 2.1 and previous work [12], [21], our ISC model can be best utilized by the applications with the following characteristics: (1) I/O intensive (not heavily CPU intensive), (2) less data access per data page (i.e., 8KB flash memory page), and (3) a smaller amount of output data after data processing. I/O intensive applications can leverage a high internal bandwidth of SSDs and the less data access per data page is closely related to the low DRAM performance inside SSDs. If the ratio of the amount of data after processing vs. that of before processing is close to 1, these applications discard ISC performance gain thereby transferring the large final output data to a host system. Based on these observations, a database application best fits for those requirements [12]. In this work, we explore another application for our ISC model: Hadoop MapReduce. We can find many Hadoop applications with I/O intensive and generally they produce a smaller amount of final outputs after processing input data. However, unfortunately, Hadoop applications cannot meet the second characteristics (less data access per data page) since they have to read every single byte of all data to process them.

Admittedly, all Hadoop applications cannot benefit from our ISC Hadoop MapReduce framework. We now explore various Hadoop applications to choose an appropriate application for our experiment and discuss the rationale of the choice.

For this Hadoop application study, we refer to Purdue MapReduce (PUMA) Benchmarks Suite [28]. PUMA benchmark suite presents a broad range of Hadoop MapReduce applications (i.e., a total of 13 applications) exhibiting various application characteristics. We run PUMA benchmark on our Hadoop machine and analyze each application for our study. After the investigation, interestingly, we found many of the applications (7 out of 13 including wordcount) can be classified into a wordcount-like application. That is to say, (1) a processed data size is a lot smaller than an original data size, (2) average per-Mapper execution time is a lot longer than the average per-Reducer execution time (i.e., Mapper intensive), (3) their final results produce counting-related values, and (4) generally CPU intensive but not highly CPU intensive computation.

For the rest of applications, particularly, Terasort can be identified as a worst Hadoop MapReduce application for our ISC framework. Sorting applications including Terasort never reduce their processed data size, which is a very unfavorable characteristic to ISC framework [12]. More critically, due to a limited DRAM resource inside our ISC devices, Terasort should generate an enormous number of flash write operations to temporarily store its intermediate data. This can critically impact NAND flash lifetime. K-means clustering is another unfavorable MapReduce application for our ISC framework because an amount of transferring data is never reduced and instead it is even increased with extra information via many iterations. Moreover, relatively high CPU computation is required to calculate the cosine-vector similarity of a given data with all the centroids.

Based on our study, we choose Hadoop wordcount as our ISC application. As we mentioned above, this wordcount is a very representative application and many other benchmark programs (term-vector, inverted-index, histogram-movies, histogram-ratings, sequence-count, ranked-inverted-index, etc) retain a very similar characteristics to the wordcount application. Thus, we expect very similar results and implications to run other applications of them. Actually, several of them use the same input data as wordcount and exhibit even almost the same execution time as the wordcount based on our experiments. We could choose other ISC-favorable applications instead of the wordcount. However, its representativeness led us to go with it. The main goal of this work is not to run various Hadoop applications, but to verify our ISC Hadoop

MapReduce framework and to explore its opportunities and challenges. Therefore, we leave implementing/running various applications including even ISC-unfavorable ones on top of our ISC framework as our future work.

4 Experimental Results

This section presents the experimental results and analyses to verify our ISC Hadoop MapReduce framework.

4.1 Evaluation Setup

4.1.1 System Configurations

For our experiments, we adopt Hadoop 0.20.205.0 as our Hadoop MapReduce framework with an input split size of 64MB by default because this has been widely adopted as a stable legacy version. Hadoop clusters consist of 5 nodes (1 namenode and 4 datanodes) with Intel i7 processor (3.40 GHz) and 8 GB memory running Ubuntu 12.04 LTS (64 bits). Each node is equipped with an ISC device of a 400GB SLC SSD with SAS 6Gb interface. This ISC device is connected to each node via a SAS Host Bus Adapter (HBA) with 6Gb, i.e., the host I/O bandwidth is 750 MB/s theoretically. The internal bandwidth of this SSD offers 3.2 GB/s, thus the bandwidth ratio is 4.27× theoretically[4]. Since ISC features inside ISC devices are enabled/disabled from the host, this ISC device is identical to a regular SSD.

For a single node setup, we install two ISC devices to the node and run 3 node instances (1 namenode and 2 datanodes) in a single machine based on our study (please refer to subsection 3.1.1). This simulates Hadoop clusters of 3 nodes for a fully distributed mode. We assign one Mapper to each ISC device and configure Hadoop with two Reducers in this single node (i.e., two Mappers and two Reducers) for our initial study. All other configurations are exactly the same as the cluster setup for fair evaluation. We employ various synthetic data which are based on PUMA benchmark and vary in size from 1GB to 16GB. We can adopt another data of a bigger size, but we found those are enough to get meaningful implication and projection to such bigger data. We run all our experiments 30 times for statistical significance and produce a value of an average.

As our baseline system, we adopt a typical Hadoop MapReduce system with SSDs running all Mappers and Reducers in the host side. That is, for fairness, we employee the exactly same system as the ISC Hadoop MapReduce system described above except only for disabling ISC features inside our SSDs. Consequently, this ISC SSDs work as typical SSDs and the host Hadoop system run all Mappers and Reducers in the host side like a typical Hadoop system.

4.1.2 Performance Metrics

We mainly measure an elapsed time (seconds) as our Hadoop system performance metrics and collect all execution time information from Hadoop JobHistoryServer thereby using *'hadoop job -history'* command.

First of all, we measure an end-to-end elapsed time. This corresponds to a total execution time of a Hadoop application and includes all setup, map, shuffle, reduce, and cleanup time. This performance represents an actual application performance where end users can be conscious of the system performance gain. To deeply analyze, we delve into Mapper because our ISC Hadoop framework offload Mapper to the ISC device and the performance gain is primarily originated from the Mapper inside our ISC devices.

An energy consumption is also another important factor. We measure the energy consumption (joule) with a Yokogawa WT330 power meter [29] as follows. Once all systems are sufficiently stabilized (i.e., idle), we measure the power in Watts (W_1). Then while running the systems, we measure the power (assuming W_2) during its execution time (t). Now energy consumption in Joules can be drawn by $(W_2 - W_1) \times t$.

4.2 Evaluation Results

4.2.1 ISC Hadoop Demonstration

We first start to exhibit a Hadoop system demonstration to compare our ISC Hadoop system with a typical Hadoop system. Figure 9 captures this demonstration. We set up both Hadoop systems on a single node with a configuration of two datanodes and one namenode as described in section 4.1.1, and run Hadoop wordcount application with same data at the same time. Both systems are connected to a SAS/SATA bus analyzer to observe an amount of data transfer from devices to a host system. As shown in the Figure, our ISC Hadoop system completes the task two times faster than a typical Hadoop system. Moreover, we clearly observe the ISC system very rarely transfers data from the device to the host. Now we make a variety of analyses of this result via extensive experiments.

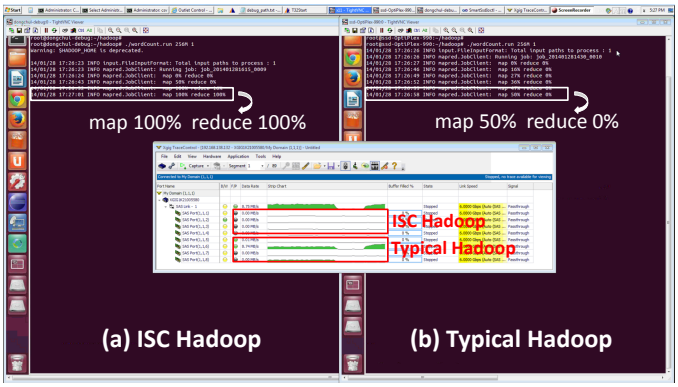


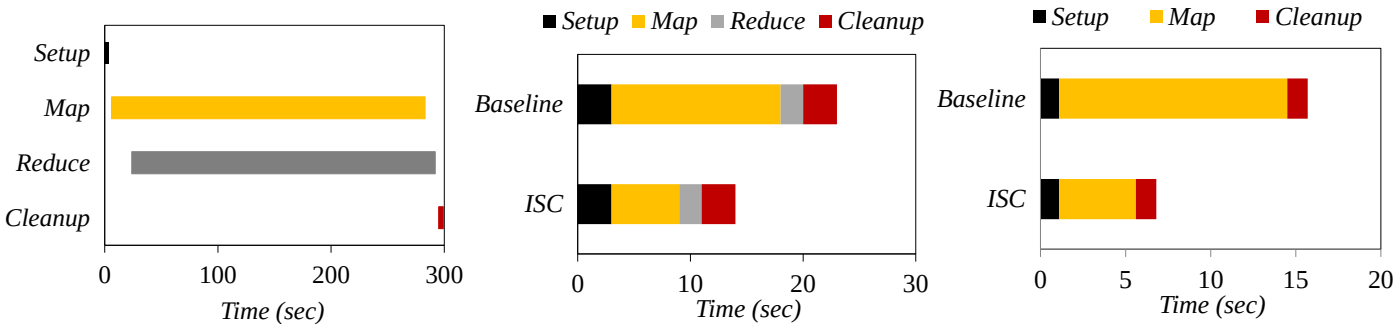
Fig. 9: Hadoop System Demonstration (ISC Hadoop vs. Typical Hadoop)

4.2.2 A Single Node

We set up two Hadoop systems for the fully distributed mode on a single node: (1) a single instance of both namenode and datanode (this corresponds to a pseudo distributed mode) and (2) a single instance of namenode and two instances of datanode.

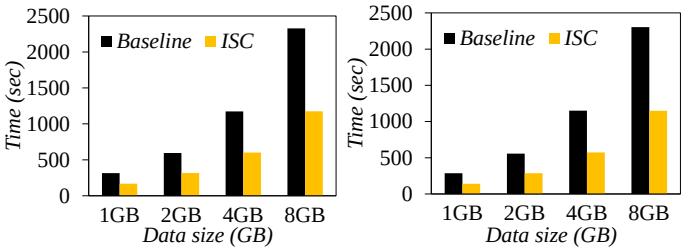
Total elapsed time: Figure 11 shows a total elapsed time on a single node with a single namenode and datanode. Our ISC Hadoop system achieves 2× faster than the typical

4. Our measured bandwidth ratio is around 3.8×, which is a little smaller than 4.27× due to the extra firmware overhead inside SSDs.



(a) Per Job (b) Per Task (c) Per Map Task

Fig. 10: Breakdown of each Hadoop execution time



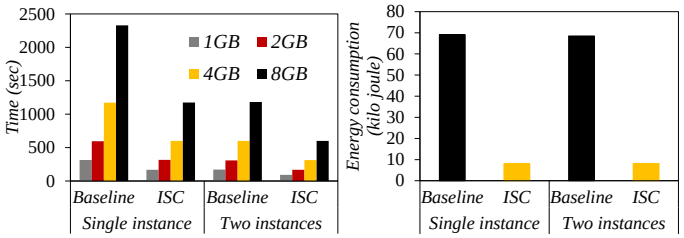
(a) Total Elapsed Time (b) Total Mapper Elapsed Time

Fig. 11: Total elapsed time on a single node (single datanode and single namenode)

Hadoop system in terms of a total elapsed time (i.e. end-to-end total execution time) (please refer to Figure 11 (a)). Figure 11 (b) displays a total elapsed time in a Mapper phase by excluding all other phases. This shows how much time each Hadoop system consumes to complete all Map tasks. Similarly, our ISC Hadoop system completes all Map tasks 2× faster than the baseline system. The results and trends in Figure 11 (b) are almost identical to those in Figure 11 (a). This is because the total execution time of all Map tasks is a significantly dominant factor of the total Hadoop running time in wordcount-like Hadoop applications (for instance, 91% with 1GB data – 98% with 8GB data of the total wordcount execution time). Figure 10 (a) demonstrates well this analysis. It breaks down the total execution time of wordcount into each Hadoop phase (baseline with 1GB data on a single node). After a short Hadoop setup time (3 seconds), the Hadoop system keeps executing each Map task (277 seconds) and completed Map tasks trigger executing a corresponding Reduce task with their output data in parallel. The total execution time of Reduce tasks (268 seconds) almost overlaps with that of Map task since it is heavily dependent on Map task completion. Lastly, Hadoop cleanup time takes about 3 seconds.

Figure 10 (b) makes a deeper analysis of the Hadoop task execution time. This figure shows the elapsed time of each Hadoop phase per each task. Each Hadoop task is subdivided into Map task and Reduce task. As a matter of fact, Hadoop sets up a job (setup time) at the beginning and deletes all intermediate data (cleanup time) after the job completion. These two tasks are performed only one time respectively throughout the whole job execution. That is, the total execution time of Hadoop application can be largely composed of one-time job setup time, many (depending on a data size) Map and Reduce tasks execution time, and one-time cleanup time. Considering this fact, Mapper execution time is a dominant factor of a total Hadoop execution time in wordcount-like Hadoop application. Based on these observations, our proposed ISC Hadoop framework moves Mapper to our ISC devices to improve Hadoop performance. As shown in Figure 10 (b), our ISC Hadoop system improves the Map task execution time by 2.5× faster (from 15 seconds to 6 seconds). All of our performance gain stems from this Mapper improvement.

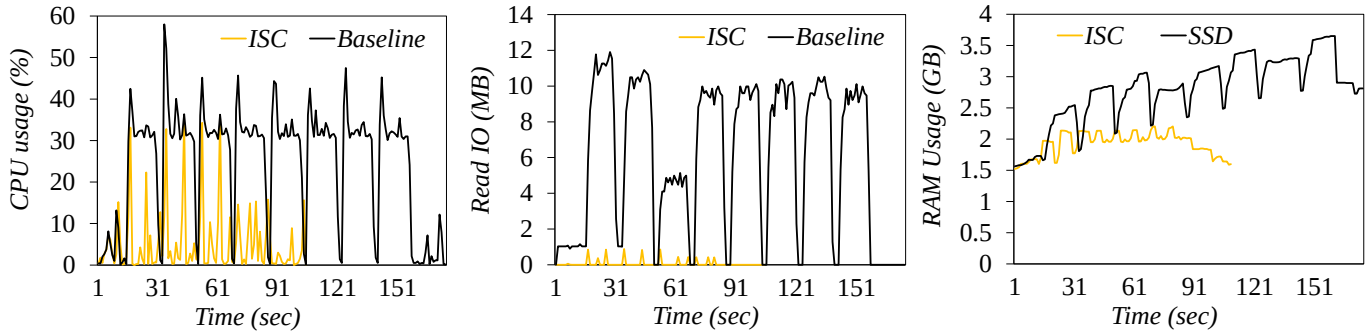
Figure 10 (c) provides a even deeper analysis of the Mapper execution time. This plot corresponds to Map task execution time (i.e., yellow bar) in the Figure 10 (b). Similarly, Map task execution time is subdivided further into three elements: Map task setup time, Map task execution time, and Map task cleanup time. Both Map task setup time and cleanup time are almost constant time (in total 1.2 seconds). Thus, if we compare a *pure* Map task execution time (i.e., yellow bar in the Figure 10) (c) between them, our performance gain is even larger (3.6×). However, each Map task requires such extra constant time (i.e., both Map task setup time and cleaning time, about 2–3 sec/Map) as well as Map task execution time. This constant time is relatively big portion (about 36%) to Map task in ISC Hadoop system, compared to the portion (14.6%) in baseline. Therefore, this larger performance gain (3.6×) is lessened to 2.5× in Figure 10 (b), and similarly to 2× in end-to-end total execution time (Figure 11).



(a) Total Elapsed Time (b) Total Energy Consumption

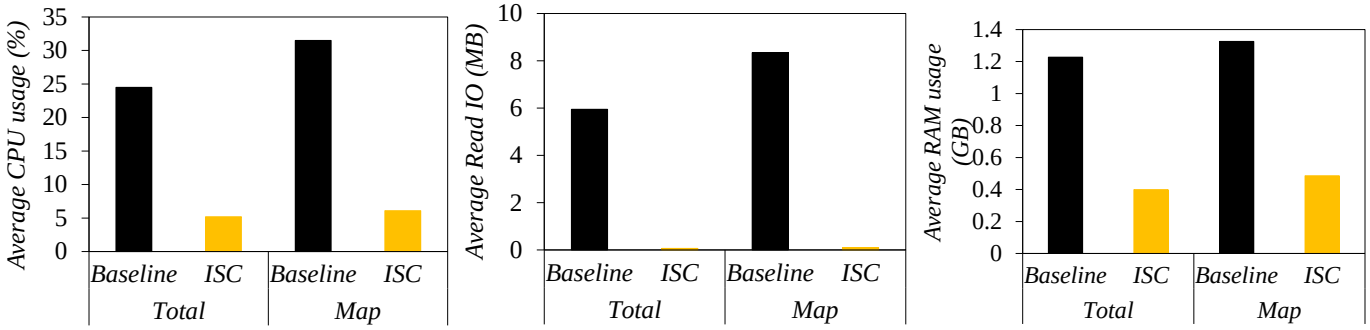
Fig. 12: Single instance vs. two instances on a single node

Now, we add one more ISC device to the host, which configures a single instance of namenode and two instances of datanode in a single node. Figure 12 (a) clearly shows 2× performance improvement with this configuration. This experiment result provides a meaningful implication that our ISC Hadoop system can offer an equivalent performance of a typical Hadoop system with the lower system cost.



(a) Host CPU Usage (b) Host Read I/O (c) Host DRAM Usage

Fig. 13: Host system resource usage



(a) Average Host CPU Usage (b) Average Host Read I/O (c) Average Host DRAM Usage

Fig. 14: Average host system resource usage

Energy consumption: Figure 12 (b) plots total energy consumption of each Hadoop system. We run the Hadoop application with 10GB data on both systems and measure energy consumption. As stated in subsection 4.1.2, we subtract the idle power (40.5W for the single instance, 45.5W for the two instances configuration) from the total power consumption. Then it is multiplied by the total execution in order to produce the total energy consumption. As shown in Figure 12 (b), our ISC Hadoop system consumes surprisingly lower energy (9×) than the typical Hadoop system with SSDs. Since we measure an additional energy consumption by eliminating an idle system power, both Hadoop systems with single ISC devices (just an SSD for the baseline system) and two ISC devices (two SSDs for the baseline system) show almost the same results. Intuitively, this is because the Hadoop system with two ISC devices (i.e., two instances of datanode) consumes two times more system power but it completes Hadoop job two times faster. Based on our experimental results, with the help of a faster execution time and lower energy consumption, our ISC Hadoop system can make a notably contribution to reduce a Total Cost of Ownership (TCO).

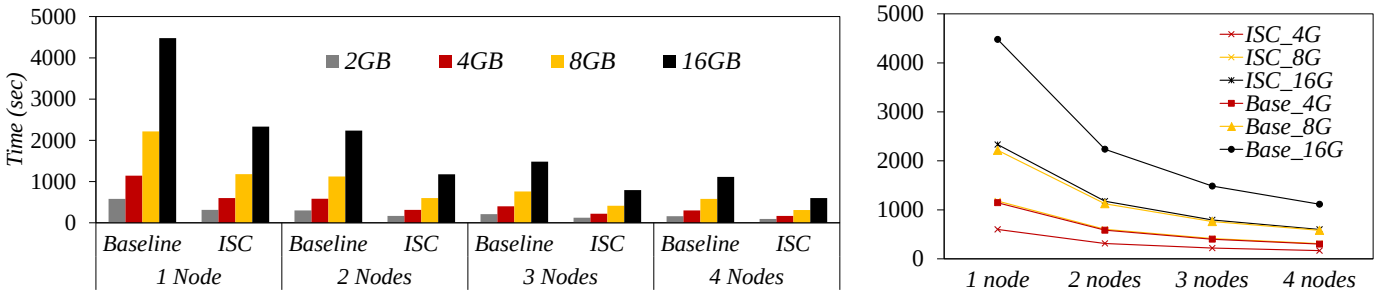
4.2.3 A System Resource Usage

All key benefits from ISC systems originates from the fact that it does not move data in storage devices to DRAM in the host system. This section clearly verifies this claim. A SATA/SAS bus analyzer is generally adopted to analyze the communication between a host system and devices in a system interface protocol level. This analyzer also provides a graphic user interface to analyze an amount of data transfer between them. As shown in the Figure 9, we have already demonstrates that ISC Hadoop system does not (or, very rarely) transfer the data from the devices to the host system.

To make a deeper analysis, we captures three main host system resources: Host CPU usage, Read I/O, and DRAM usage. For this, we set up two Hadoop systems (i.e., ISC and baseline) in a respective single machine with two SSDs (or ISC devices) and run a Hadoop application with 1GB data. We observe each system resource over the execution time (Figure 13) and an average resource resource usage respectively (Figure 14). Moreover, we measure each average value not only during total execution time (refer to Total) but also only Map task execution periods (refer to Map). Interestingly, we observe 8 repetitive ridges in the Figure 13. This is because we use 1GB data and configure two datanode instances in a single machine where one Mapper is assigned to each datanode for an accurate analysis. Each datanode executes its own Map task by loading respective input split data of 64MB (that is, 128MB for two Map tasks at the same time). Therefore, we can see 8 separate sections (1GB = 128MB × 8).

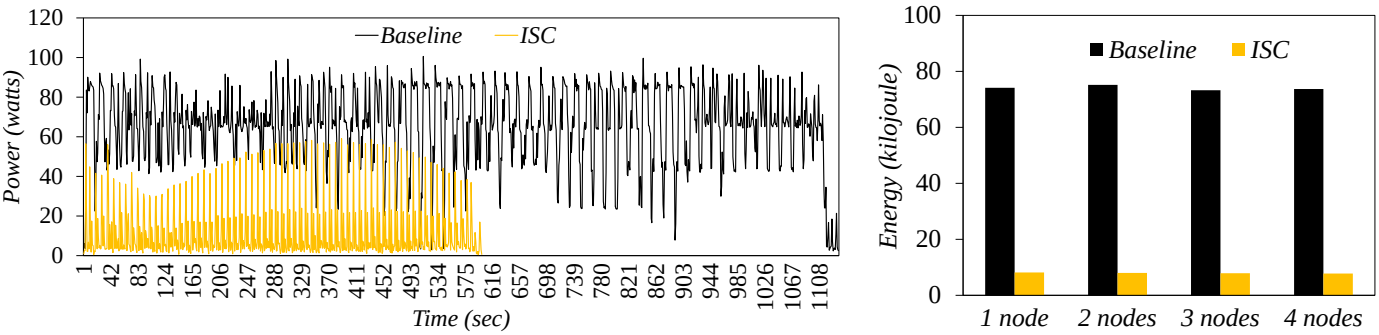
Host CPU Usage: Figure 13 (a) and Figure 14 (a) show the host CPU usage during Hadoop application execution time. Both results demonstrate our ISC Hadoop system consumes even less host CPU resources (3.7× less on total average and 4.2× less for Map task execution periods). This is because the ISC Hadoop system executes all Map tasks inside the ISC devices thereby using their own CPUs and does not transfer data to the host system for Map task computation. We can observe that intuitively a more CPU resource is consumed for a period of Map task computation.

Host Read I/O: Unlike a conventional CPU-centric system loading all data to the host for computation, an ISC



(a) Total Elapsed Time (b) Total Elapsed Time

Fig. 15: Total elapsed time on Hadoop clusters with a various number of datanode. A node here means datanode.



(a) Power Consumption Over Time (4 datanodes) (b) Total Energy Consumption

Fig. 16: Power and energy consumption on Hadoop clusters with 16GB data

system does not need to move data from a device to the host. This can significantly reduce Host I/O between them. Both Figure 13 (b) and Figure 14 (b) support this claim. A typical Hadoop system generates lots of read I/Os to load all data from devices to the host, while our ISC Hadoop system very rarely generates read I/Os (almost 2 orders of magnitude less). Especially for the period of each Map task computation, the typical Hadoop system keeps reading data from the devices. This is also verified by our SATA/SAS bus analyzer (Figure 9). We capture both read and write I/Os, but write I/Os are almost ignorable (less than 1%) for this application. Thus, we exhibit only read I/Os in the plots. In addition, we also verified that when we add up all amounts of read I/Os, it is almost equivalent to the input data size of 1GB.

Host DRAM Usage: Host DRAM is another important factor to promote ISC systems. Typically, all data loaded from devices are first stored in host DRAM for computation, which results in a more memory consumption. To measure this host memory consumption, we do not run any other applications after reboot and flush all data/page caches. Thus, both starting points of the plot are almost the same. After running the Hadoop application, the baseline system keeps consuming DRAM until all Map tasks completes. On the other hand, our ISC Hadoop system consumes even less DRAM (3× less). This has a critical impact on energy consumption since it has been widely known that DRAM takes 25% – 40% of a system power consumption [30], [31], [32], [33].

4.2.4 Hadoop Clusters

We set up clusters of 5 nodes (1 namenode and 4 datanodes) to verify whether or not all implications from a single node can be applied to real Hadoop clusters. Each datanode is equipped with a ISC device and connects to each others via 1Gb Ethernet switch. We run the same Hadoop application and data varying 1GB to 16GB (we eliminate results with 1GB in the plots).

Figure 15 (a) shows the total elapsed time on the clusters. The results are very similar to those on a single node: our ISC Hadoop system is 2× faster than baseline Hadoop system, which implies that ISC Hadoop clusters of a half size can achieve a similar performance to the baseline Hadoop clusters, or that ISC Hadoop clusters of the same size as the baseline Hadoop clusters can process two times amount of data for the same period of time. Figure 15 (b) illustrates well this claim. For instance, the plot of the ISC Hadoop system with 16GB data (ISC_16G) is almost overlapped with the plot of the baseline Hadoop system with 8GB data (Base_8G). We do not display the total Mapper elapsed time since the results are not so much different from the Figure 15. This is already verified in the Figure 11.

As we analyzed in subsection 4.2.2, the total energy consumption of each size of clusters is almost identical (about 9.3× lower) since bigger clusters complete the same task with a shorter time (Figure 16 (b)). Note: like a sinle node experiment, we also eliminate an idle power. Figure 16 (a) plots a power consumption over time in the clusters of 4 datanodes (plus 1 namenode) with 16GB data. This shows that the ISC Hadoop system consumes about 4.6× lower power than the baseline system, which induces over 9× lower energy consumption considering 2× faster execution time.

5 Related Work

Some studies claim the main idea of offloading a computation into storage devices originates from CASSM or

Data Base Computer (DBC) in 1970s [34], [35]. They adopt process-per-track or process-per-head architecture which associates a processing logic with each read/write head of a hard disk drive. As a matter of fact, both Active Disks and iDisks in 1998 start to explore a modern In-Storage Computing (also called In-Storage Processing or In-situ Processing) design [14], [15] with the help of the advance of HDD technologies (i.e., bandwidth growth and cost dropping). They try to offload key application functions such as query operators inside HDDs to reduce data movement, which corresponds to a current In-Storage Computing model. However, still low computing capabilities of HDDs prevent HDD-based ISC from successful landing to practical systems.

However, the advent of SSDs paves the way for people to rethink of In-Storage Computing as a practical and promising computing model even in industries as well as in the academia. Samsung recently promotes Storage Intelligence for In-Storage Computing and Multi-Stream to improve SSD performance and better NAND flash endurance, which is adopted by storage companies like Pure Storage [36]. IBM applies ISC to their Blue Gene supercomputers to leverage the high internal bandwidth of SSDs [37]. Oracle's Exadata also starts to offload complex database processing into their storage servers [38].

SSD-based ISC attracts academia as well. Logothetis et al. propose iMR for in-situ log processing [39]. It suggests an architecture with a prototype system based on a best-effort distributed stream processor. Kim et al. try to move a scan operator in database to SSDs based on simulation [40] and later Do et al. integrate Microsoft SQL Server with real SSDs thereby offloading database key operators to the SSDs [12]. Kang et al. propose Hadoop MapReduce framework for ISC [21]. This work addresses same topic as ours; however, it supports only one node with just limited functions. On the other hands, our proposed ISC Hadoop MapReduce framework fully follows the exiting MapReduce framework and can operates on distributed Hadoop clusters. Boboila et al. propose Active Flash to apply ISC to data analytics [41] and later Tiwari et al. develop more on this Active Flash for extreme scale machine based on OpenSSD platform [10], [42]. Seshadri et al. propose a Willow system [43]. This is a user-programmable SSD prototype system allowing programmers to augment and extend the semantics of an SSD with application-specific features.

Our work explore the opportunities and challenges of Hadoop MapReduce framework based on In-Storage Computing model.

6 Conclusion

Unlike the traditional CPU-centric computing model, SSD-based In-Storage Computing (ISC) is a new computing paradigm that enables SSDs to play a major role in computation, not just in yet-another-faster HDDs. This SSD-based ISC system offloads main features from a host to an SSD to make the best use of SSD potential capabilities such as its high interval bandwidth and computational power with low power CPUs.

In this paper, we apply the ISC model to the Hadoop MapReduce framework–a de facto standard distributed computing framework for big data processing. After investigation, we offload Mapper from a host Hodoop system to SSDs and integrate the existing Hadoop MapReduce system with our ISC devices. This system co-design gives rise to the following challenging technical issues: (1) discrepancy in data representation and (2)system interfaces between a host system and a ISC device, (3) Hadoop data split, and (4) feature offloading. To address the data representation and system interface issues, we add a software layer between the host Hadoop framework and our ISC devices. This software interface layer is in charge of converting a file name to LBA information and communicating Java codes with C/C++ codes. The data split issue in HDFS input split data of 64MB (default) is another critical issue. To resolve this, we consult a host Hadoop system to receive the split information and store the split part as a separate data file. Then our ISC Hadoop system process the separate file as new input data. Which part should be offloaded to the ISC device is always a big issue for ISC research and implementation because moving all features to SSDs does not necessarily result in a performance improvement. A fully distributed mode on a single Hadoop machine is yet-another interesting issue for Hadoop MapReduce research work. Although Hadoop does not support this mode, it can provide a very useful and efficient way by simulating the fully distributed Hadoop clusters only in a single machine. All of our initial studies and analyses have been made on this mode and it can be applied to the real multi-node Hadoop clusters.

Our extensive experiments and results demonstrate that our ISC Hadoop system exhibits a siginficant performance improvement (2× faster) as well as marvelous energy saving (more than 9× less) than a typical Hadoop system. Moreover, our deep analyses are provided to delve into our performance gains.

As we mentioned, we classify wordcount-like Hadoop applications as ISC-favorable applications and choose Hadoop wordcount for our experiments. We expect all such applications will show very similar results and remain its verification via a system implementation as our future work. In addition, we can also investigate ISC-nonfavorable Hadoop applications in the future.

References

[1] H. Kim and S. Ahn, " BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage ," in *FAST*, 2008.

[2] D. Park, B. Debnath, and D. Du, "CFTL: A Convertible Flash Translation Layer Adaptive to Data Access Patterns," in *SIGMETRICS*, 2010, pp. 365–366.

[3] A. Caulfield, L. Grupp, and S. Swanson, "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications," in *ASPLOS*, 2009.

[4] Samsung, "Why ssds are awesome: An ssd primer," Samsung Electronics, Tech. Rep., 2014. [Online]. Available: http://www.samsung.com/global/business/semiconductor/minisite/SSD/global/html/about/whitepaper01.html

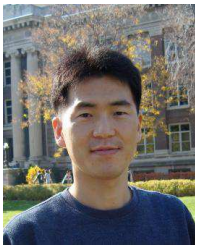
[5] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *Micro, IEEE*, vol. 34, no. 4, pp. 36–42, 2014.

[6] Hortonworks, "What is hadoop?" http://hortonworks.com/hadoop/mapreduce/.

[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *OSDI*, 2004.

[8] T. White, *Hadoop: The Definitive Guide, Third Edition*. OReilly, 2012.

[9] J. Gray, "Distributed computing economics," Microsoft Research, Tech. Rep. MSR-TR-2003-24, March 2003. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=70001
[10] D. Tiwari, S. Boboila, S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin, "Active Flash: Towards Energy-Efficient, In-Situ Data Analytics on Extreme-Scale Machines," in *FAST*, 2013.
[11] nVidia Corporations, "Cuda programming," http://www.nvidia.com/object/cuda_home_new.html.
[12] 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 *SIGMOD*, 2013, pp. 1221–1230.
[13] JDSU, "Xgig 1000 12gbps sas analyzer," http://www.jdsu.com/ProductLiterature/xgig1000-12g-analyzer-ds-san-tm-ae.pdf.
[14] K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A case for intelligent disks (idisks)," *SIGMOD Rec.*, vol. 27, no. 3, pp. 42–52, 19980.
[15] A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," in *ASPLOS*, 1998, pp. 81–91.
[16] A. De, M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating data analysis in next-generation ssds," in *FCCM*, 2013, pp. 9–16.
[17] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *Journal of System Architecture*, vol. 55, no. 5-6, pp. 332–343, 2009.
[18] D. Park, B. Debnath, and D. Du, "A Workload-Aware Adaptive Hybrid Flash Translation Layer with an Efficient Caching Strategy," in *MASCOTS*, 2011, pp. 248 – 255.
[19] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, 2010.
[20] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP*, 2003.
[21] Y. Kang, Y. suk Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart ssd," in *MSST 13*, 2013.
[22] Seagate, "Seagate kinetic hdds," http://www.seagate.com/products/enterprise-servers-storage/nearline-storage/kinetic-hdd/.
[23] HGST, "Open ethernet drive," http://www.hgst.com/science-of-storage/emerging-technologies/open-ethernet-drive-architecture.
[24] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX ATC*, 2008, pp. 57–70.
[25] S. Lee and B. Moon, "Design of Flash-based DBMS: An In-page Logging Approach," in *SIGMOD*, 2007.
[26] D. Park and D. Du, "Hot Data Identification for Flash-based Storage Systems using Multiple Bloom Filters," in *MSST*, 2011, pp. 1 – 11.
[27] D. Park, B. Debnath, Y. J. Nam, D. H. Du, Y. Kim, and Y. Kim, "An on-line hot data identification for flash-based storage using sampling mechanism," *ACM Applied Computing Review*, vol. 13, no. 1, pp. 51–64, 2013.
[28] F. Ahmad, S. Lee, M. Thottethodi, and T. N. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite," Purdue University, Tech. Rep. TR-ECE-12-11, October 2012.
[29] Yokogawa, "Yokogawa wt300 series digital power meter," http://tmi.yokogawa.com/files/uploaded/WT300Series_Final.pdf.
[30] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *ISCA*, 2010.
[31] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," 2008.
[32] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," 2009.
[33] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *ISCA*, 2009.
[34] S. Y. W. Su and G. J. Lipovski, "Cassm: A cellular system for very large data bases," in *VLDB*, 1975, pp. 456–472.
[35] K. Kannan, "The design of a mass memory for a database computer," in *ISCA*, 1978, pp. 44–51.
[36] Pure Storage, "Pure storage flasharray," http://www.purestorage.com.
[37] Jülich Research Center, "Blue gene active storage boosts i/o performance at jsc," http://cacm.acm.org/news/169841-blue-gene-active-storage-boosts-i-o-performance-at-jsc, 2013.
[38] Oracle Corporation, "Oracle exadata white paper," 2010.
[39] D. Logothetis, C. Trezzo, K. C. Webb, and K. Yocum, "In-situ mapreduce for log processing," in *USENIX ATC*, 2011.
[40] S. Kim, H. Oh, C. Park, S. Cho, and S. Lee, "Fast, energy efficient scan inside flash memory," in *ADMS*, 2011, pp. 36–43.
[41] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman, "Active flash: Out-of-core data analytics on flash storage," 2012.
[42] OpenSSD project, "Jasmine openssd platform," http://www.openssd-project.org/wiki/Jasmine_OpenSSD_Platform.
[43] S. Seshadri, M. Gahagan, S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, "Willow: A user-programmable ssd," in *OSDI*, 2014, pp. 67–80.



Dongchul Park is currently a research scientist in System Architecture Lab. (SAL) at Samsung R&D Center in San Jose, California. He received his Ph.D. and M.S. degrees in Computer Science and Engineering at the University of Minnesota, Twin Cities in 2012 and 2006 respectively. His research interests lie in storage system design and applications including in-storage computing, big data processing, Hadoop MapReduce, data deduplication, key-value store, non-volatile memories (Flash-based SSDs), hot/cold data identification, embedded systems, and shingled magnetic recording disks.

Kwanghyun Park is a research staff member at NEC Laboratories America, Princeton, New Jersey. He received the Ph.D. degree in Electrical and Computer Engineering from the University of Minnesota, Twin Cities in 2010, and the and B.S. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Bangladesh, in 2002. His research interests include Non-volatile memory, data deduplication, key-value store, and storage systems.

Yang-Suk Kee (Fellow, IEEE) is currently the Qwest Chair Professor in Computer Science and Engineering Department, University of Minnesota, Twin Cities. He received the B.S. degree in mathematics from National Tsing-Hua University, Taiwan, R.O.C. in 1974, and the M.S. and Ph.D. degrees in computer science from the University of Washington, Seattle, in 1980 and 1981, respectively. His research interests include storage systems, cyber security, sensor networks, multimedia computing, and high-speed networking.

Jignesh M. Patel (Fellow, IEEE) is currently the Qwest Chair Professor in Computer Science and Engineering Department, University of Minnesota, Twin Cities. He received the B.S. degree in mathematics from National Tsing-Hua University, Taiwan, R.O.C. in 1974, and the M.S. and Ph.D. degrees in computer science from the University of Washington, Seattle, in 1980 and 1981, respectively. His research interests include storage systems, cyber security, sensor networks, multimedia computing, and high-speed networking.