A Virtualization-Based Hybrid Storage System for a Map-Reduce Framework

Aseffa, DEREJE TEKILU[a†], Nonmember and Chin-Hsien WU[b†], Member

SUMMARY A map-reduce framework is popular for big data analysis. In the typical map-reduce framework, both master node and worker nodes can use hard-disk drives (HDDs) as local disks for the map-reduce computation. However, because of the inherit mechanical problems of HDDs, the I/O performance is a bottleneck for the map-reduce framework when I/O-intensive applications (e.g., sorting) are performed. Replacing HDDs with solid-state drives (SSDs) is not economical, although SSDs have better performance than HDDs. In this paper, we propose a virtualization-based hybrid storage system for the map-reduce framework. The objective of the paper is to combine the advantages of the fast access property of SSDs and the low cost of HDDs by realizing an economical design and improving I/O performance of a map-reduce framework in a virtualization environment. We propose three storage combinations: SSD-based, HDD-based, and a hybrid of SSD-based and HDD-based storage systems which balances speed, capacity, and lifetime. According to experiments, the hybrid of SSD-based and HDD-based storage systems offers superior performance and economy.

key words: virtualization, hybrid storage systems, map-reduce, solid-state drives

1. Introduction

Big Data is a type of data that have grown exponentially and require a powerful data computing model. The big-data applications focus on computation, storage, and retrieval of a large amount of unstructured, semi-structured, and even structured data. A map-reduce framework can be used to analyze and compute big data. The map-reduce framework employs parallel processing and distributed algorithms to analyze a large scale of data sets on a cluster of commodity machines [1]. A typical cluster of commodity machines usually uses hard-disk drives (HDDs) as local disks. Because of the inherit mechanical problems, HDDs remain an I/O performance bottleneck for I/O-intensive applications (e.g., sorting). In order to alleviate the I/O performance bottleneck, the previous studies [2]–[4] show that replacing HDDs with solid-state drives (SSDs) can significantly improve the performance of the map-reduce computation of I/O-intensive applications due to the fast access property of SSDs. However, replacing HDDs with SSDs is not economical, because SSDs are more expensive than HDDs. Therefore, we investigate three storage combinations: SSD-based, HDD-based, and the hybrid of SSD-based and HDD-based storage systems for the map-reduce framework to have an efficient and economical design.

In this paper, we propose a virtualization-based hybrid storage system for a map-reduce framework. A virtualization-based hybrid storage system for the map-reduce framework is possible when I/O-intensive applications are run in a virtualization environment by a powerful computing system with a global address space, when we rethink the research work [16]–[18]. Therefore, the objective of the proposed method is to combine the advantages of the fast access property of SSDs and the low cost of HDDs by realizing an economical design and improving I/O performance of a map-reduce framework in a virtualization environment. In order to realize an economical design, we should extend the lifetime of the expensive SSDs by considering the actual total amount of data written as the lifetime issue. In addition to improving I/O performance, we take account of the speed of read and write operations of an SSD or an HDD. Because each virtual machine’s storage capacity is fixed in advance, it needs additional overhead to adjust the storage capacity when the available free space is not enough. Consequently, we take account of the available free space of an SSD or an HDD. Overall, we think the three factors (i.e., speed, capacity, and lifetime) should be considered for the objective of the proposed method. In the proposed map-reduce framework, assume that there are one master node and (S + H) worker nodes, where S worker nodes are equipped with SSDs and H worker nodes are equipped with HDDs. When a map-reduce application is executed, the (S + H) worker nodes will perform the computation, storage, and retrieval of a large amount of data in the map and reduce phase. Therefore, we propose a weighting function by considering the characteristics of HDDs (e.g., speed and capacity) and SSDs (e.g., speed, capacity, and lifetime) to improve the map and reduce task execution time. Because of the combinations of (S + H) worker nodes, three storage combinations: SSD-based, HDD-based, and the hybrid of SSD-based and HDD-based storage systems are proposed and adopt the weighting function to assign the appropriate tasks to the powerful or suitable worker nodes. The results of experiments show that the proposed method can have better performance than HDD-based storage systems. In addition, because we use the hybrid of SSD-based and HDD-based storage systems, the proposed method is obviously more economical than SSD-based storage systems. As a result, the hybrid of SSD-based and HDD-based storage sys-
tems offers superior performance and economy.

The rest of this paper is organized as follows: In Sect. 2, we briefly introduce HDDs/SSDs and a map-reduce framework. Also, we briefly review related work in Sect. 2. In Sect. 3, we explain the motivation. In Sect. 4, we present a virtualization-based hybrid storage system for a map-reduce framework. In Sect. 5, we present the experimental setup and the experimental results. Finally, we provide the concluding remarks in Sect. 6.

2. Background Knowledge

2.1 HDDs vs. SSDs

A hard-disk drive (HDD) is a data storage that uses rotating disks coated with magnetic material for storing and retrieving data [7]. An HDD consists of a number of rotating magnetic disks and has a read/write head to access the magnetic disks. When read/write operations are required for an HDD, the read/write head will move to a specific location to access data by rotating magnetic disks. Therefore, operations on HDDs could cause mechanical latency that includes rotational latency and seek time (e.g., milliseconds) [8]. The read/write head could spend seek time to find a specific track on the rotating disks. The read/write head could also spend rotational latency to find a specific sector on the track after the seek operation is performed. I/O performance based on HDDs is always slow due to the mechanical latency. When I/O-intensive applications are performed on HDDs, the I/O performance could become a bottleneck.

A solid-state drive (SSD) is an alternative data storage to replace traditional HDDs because of its breakthrough access performance. Because SSDs mainly adopt NAND flash memory as storage area, SSDs can access data on integrated circuits [9] without the mechanical latency (e.g., rotational latency and seek time). Furthermore, SSDs have attractive advantages such as small size, fast access speed, shock resistance, light weight, and low-power consumption [10]. Therefore, SSDs can be used to improve the I/O performance and alleviate the I/O bottleneck caused by HDDs. However, SSDs are more expensive than HDDs, when we compare their cost. As PC magazine post of February 18, 2015, for the same capacity and form factor 1TB internal 2.5-inch drive, it costs about USD60 to USD75 for an HDD and doubles for an SSD to USD130 to USD150. That translates into 7 cents per gigabyte for the HDD and 14 cents per gigabyte for the SSD [11].

2.2 A Map-Reduce Framework

A map-reduce framework is a popular computation model [11], [6] because of its simplicity, elastic scalability, and fault tolerance. The map-reduce framework can be used to analyze and compute a massive amount of data in parallel. For example, Hadoop is an open source software that can perform the map and reduce computation on a large cluster of commodity machines. The map and reduce computation takes a large input data and splits the data into a number of chunks whose fixed size ranges from 16MB to 128MB. Each chunk (i.e., split) can be processed by the map and reduce computation. The map and reduce computation uses two types of functions: map and reduce. The map function takes each chunk as input data and parse the input data into a set of key/value pairs. The reduce function takes the set of key/value pairs as input data and reduces them to the final results. The map and reduce computation can be run on a large cluster of commodity machines. Each machine in the cluster is called a node. The cluster consists of two node types: a master node and a worker node. In the cluster, the master node is usually only one and others are worker nodes. The master node needs to manage the overall execution of the map and reduce computation. For example, the master node is responsible to assign and schedule the map and reduce computation to worker nodes, monitor the computation, and restart the computation on failure. The worker node that performs the map function is called a map worker node and the worker node that performs the reduce function is called a reduce worker node.

2.3 Related Work

In recent years, the performance improvement of the map-reduces framework has become important. The work in [3] revealed that SSD can significantly lessen I/O bottleneck for a Hadoop cluster. In [2], authors proposed that the replacement of HDDs with SSDs is beneficial and reduces power consumption, especially for I/O-intensive map-reduce workloads. In [5], authors explored cost-effective map-reduce configurations and showed that the hybrid SSD/HDD configuration performs between whole HDD configuration and whole SSD configuration with varying size of DRAM. They investigated proper use of DRAM or SSD for intermediate data to accelerate the map-reduce performance. The work in [4] revealed that SSDs deliver a significant map-reduce performance improvement, when compared to HDDs of equal aggregate I/O bandwidth. They also investigated that adding SSDs to the existing HDD-based cluster can improve performance if the space of SSDs can be properly allocated to multiple HDFS and shuffle directories. As a result, in this paper, we fully investigate and analyze three storage combinations: SSD-based, HDD-based, and the hybrid of SSD-based and HDD-based storage systems that balance speed, capacity, and lifetime of HDDs and SSDs for a map-reduce framework in a virtualization environment. We also propose a weighting function to properly and efficiently dispatch the map-reduce computation among possible storage combinations for a map-reduce framework.

In [22], authors revealed that RAID is a storage technology that combines multiple storage devices into a single logical unit for the purpose of data redundancy or performance improvement. In fact, RAID technology can be used in the proposed method by adopting RAID-based SSDs or RAID-based HDDs to replace one single SSD or one single
HDD. Even we adopt RAID-based SSDs or RAID-based HDDs, the characteristics of SSDs and HDDs should be considered in the paper.

In [19], authors proposed Hystor that is a hybrid storage system that achieves its optimization objectives of data management. Hystor can identify performance-critical blocks and semantically-critical blocks (e.g. file system metadata). Then, Hystor offers the blocks a high priority to stay in the SSD to improve system performance. Furthermore, Hystor can buffer incoming writes into the low-latency SSD for improving performance of write-intensive workloads. In [23], authors proposed Apple Fusion Drive that is a hybrid storage system that combines HDDs with SSDs, and presents it as a single logical unit. OSs should automatically manage the contents of the hybrid storage system so that most frequently accessed files are stored on faster SSDs, while infrequently accessed items move to stay on HDDs. The main strategy of Hystor and Apple Fusion Drive is to put important data or frequently used data on faster SSDs to improve system performance. However, when a map-reduce application is run in a virtualization-based hybrid storage system, the map and reduce tasks are all important and accessed. Even the hybrid storage system is used, the proposed method still needs to assign and schedule tasks to the appropriate worker nodes by considering the characteristics of SSDs and HDDs.

In [20], authors proposed Spark that is an in-memory distributed computing framework. In [21], authors proposed Dyrad that is a general-purpose distributed execution engine for coarse-grain data-parallel applications. In [24], authors proposed Apache Tez that is an open-source framework designed to build data-flow driven processing runtime. In-memory distributed computing frameworks can have a good performance if data can fit into main memory to be processed. However, if data cannot fit into main memory, in-memory distributed computing frameworks also use external storage devices for data access and could cause performance degradation. Therefore, the proposed method still needs to consider the characteristics of SSDs and HDDs.

3. Motivation

In the typical map-reduce framework, the map and reduce computation can be run on a cluster of the master and worker nodes. The local I/O storage devices are usually HDDs for the master node and worker nodes. Because of inherit mechanical problems of HDDs, the I/O performance is a bottleneck for the map-reduce framework when compared to the performance of main memory and CPU cores. To overcome the challenge, we use the hybrid storage systems of SSDs and HDDs as local I/O storage devices for the map-reduce framework. In the proposed map-reduce framework, we use a cluster that consists of one master node and \((S + H)\) worker nodes, where \(S\) worker nodes are equipped with SSDs and \(H\) worker nodes are equipped with HDDs. The objective of the proposed method is to combine the advantages of the fast access property of SSDs and the low cost of HDDs by realizing an economical design and improving I/O performance of a map-reduce framework in a virtualization environment. In order to realize an economical design, we should extend the lifetime of the expensive SSDs by considering actual total amount of data written as the lifetime issue. In order to improve I/O performance, we take account of the speed of read and write operations of an SSD or an HDD. Because each virtual machine’s storage capacity is fixed in advance, it needs additional overhead to adjust the storage capacity when the available free space is not enough. Consequently, we take account of the available free space of an SSD or an HDD. Overall, we think the three factors (i.e., \(\text{Speed} \times \text{Capacity} \times \text{Lifetime}\)) should be considered for the objective of the proposed method. However, it is difficult to define the best balance in actual systems, the weighting ratio method can be defined such as \((x \times \text{Speed} + y \times \text{Capacity} + z \times \text{Lifetime})\), where \(x + y + z = 1\) and \(0 < x, y, z < 1\). Four considerations are listed in the following:

- Consideration 1: If developers prefer the high speed of the storage devices, the value of \(x\) should be larger than \(y\) and \(z\).
- Consideration 2: If developers prefer the balance capacity of the storage devices, the value of \(y\) should be larger than \(x\) and \(z\).
- Consideration 3: If developers prefer the longer lifetime of the SSDs, the value of \(z\) should be larger than \(x\) and \(y\).
- Consideration 4: If developers think the importance of the three factors is the same, the values of \(x\), \(y\), and \(z\) should be the same.

We suggest that developers can use the 4 considerations to find the most preferred balance in the systems. However, Consideration 4 is the objective of the paper.

4. A Virtualization-Based Hybrid Storage System for a Map-Reduce Framework

4.1 Overview

The proposed method is to handle I/O-intensive applications that are run in a virtualization environment by a powerful computing system with a global address space. According to the architecture design [18], the single computing system can run some virtual machines with a logically shared storage system. In fact, the logically shared storage system is composed of distributed storage devices connected by high-performance and extremely low-latency storage network in a single rack. Because the issue of data locality in a single rack is not so serious, we propose a definition of a virtualization-based hybrid storage system for a map-reduce framework as follows:

**Definition:** There are one master node and \((S + H)\) worker nodes in a physical computing system, where \(S\) worker nodes are equipped with SSDs and \(H\) worker nodes are
equipped with HDDs. Each worker node could be performed by a virtual machine in the physical computing system. When a map-reduce application is executed in the physical computing system, it requires $M$ map tasks to perform the map function and $R$ reduce tasks to perform the reduce function. Based on the general definition, we propose how to assign $(M + R)$ tasks to $(S + H)$ worker nodes in terms of characteristics of SSDs and HDDs. As shown in Fig. 1, the input data is partitioned into a set of $M$ map tasks (i.e., splits). Then, we can use $(S + H)$ worker nodes to handle the $M$ map tasks. After the map phase, $R$ reduce tasks are produced, and we can also use $(S + H)$ worker nodes to handle the $R$ reduce tasks. After the reduce phases, final results are produced.

4.2 Map Phase

When the input data is partitioned into a set of $M$ splits, we need to assign $M$ map tasks to $(S + H)$ worker nodes. There are three cases that need to consider:

4.2.1 Case I: All $M$ Map Tasks Are Performed by $S$ Map Worker Nodes with SSDs

- If $(M \leq S)$, we can choose $M$ map worker nodes with SSDs according to the $SCL$ weighting function (i.e., $Speed \times Capacity \times Lifetime$). The $SCL$ weighting function is $Speed \times Capacity \times Lifetime$, where $Speed$ denotes the average speed of read and write operations of an SSD, $Capacity$ denotes the available free space of an SSD, and $Lifetime$ denotes the maximum allowable amount of data written to an SSD divided to the actual total amount of data written to the SSD. Because SSDs are expensive devices and its lifetime could be affected by a large amount of data written due to the characteristics of NAND flash memory, we consider the actual total amount of data written as the lifetime issue. The dispatching rule is to dispatch each map task to the map worker node with the largest $SCL$ weighting value. We list the pseudo code in the following Algorithm 1:

- If $(M > S)$, we can schedule $M$ map tasks on $S$ map worker nodes with SSDs according to the following scheduling rule: Assume that $s_i$ denotes a map worker node whose $SCL$ weighting value is $Speed_i \times Capacity_i \times Lifetime_i$. When $M$ map tasks are dispatched, $s_i$ can get $M \times \frac{Speed_i \times Capacity_i \times Lifetime_i}{\sum_{j=1}^{SCL} Speed_j \times Capacity_j \times Lifetime_j}$.
map tasks.

Algorithm 1 Case I for \((M \leq S)\)
\[
\begin{align*}
1: & \text{ Let } SCL \text{ be } S \text{ speed } \times \text{Capacity } \times \text{Lifetime} \\
2: & \text{ Calculate each map worker node’s } SCL \text{ weighting value} \\
3: & \text{ for } i = 1 \text{ to } M \text{ do} \\
4: & \quad \text{ Select one map worker node with the largest } SCL \text{ weighting value} \\
5: & \quad t_i^{m} \text{ is dispatched to the map worker node} \\
6: & \quad \text{ Recalculate the map worker node’s } SCL \text{ weighting value} \\
7: & \end{end}
\]

4.2.2 Case II: All M Map Tasks Are Performed by H Map Worker Nodes with HDDs

- If \((M \leq H)\), we can choose \(M\) map worker nodes with HDDs according to the SC weighting function (i.e., \(S \text{ speed } \times \text{Capacity}\)), where \(S \text{ speed}\) denotes the average speed of read and write operations of an HDD, and \(\text{Capacity}\) denotes the available free space of an HDD. The dispatching rule is to dispatch each map task to the map worker node with the largest \(SCL\) weighting value. We list the pseudo code in the following Algorithm 2:

- If \((M > H)\), we can schedule \(M\) map tasks on \(H\) map worker nodes with HDDs according to the following scheduling rule: Assume that \(h_i\) denotes a map worker node, whose \(S\) weighting value is \(S \text{ speed }_i \times \text{Capacity }_j\). When \(M\) map tasks are dispatched, \(h_i\) can get this much \([M \times \frac{S \text{ speed }_i \times \text{Capacity }_j}{\sum_j (S \text{ speed }_j \times \text{Capacity }_j)}]\) map tasks.

Algorithm 2 Case II for \((M \leq H)\)
\[
\begin{align*}
1: & \text{ Let } SC \text{ be } S \text{ speed } \times \text{Capacity} \\
2: & \text{ Calculate each map worker node’s } SC \text{ weighting value} \\
3: & \text{ for } i = 1 \text{ to } M \text{ do} \\
4: & \quad \text{ Select one map worker node with the largest } SC \text{ weighting value} \\
5: & \quad t_i^{m} \text{ is dispatched to the map worker node} \\
6: & \quad \text{ Recalculate the map worker node’s } SC \text{ weighting value} \\
7: & \end{end}
\]

4.2.3 Case III: Part of M Map Tasks Is Performed by S Map Worker Nodes with SSDs and the Rest Is Performed by H Map Worker Nodes with HDDs

We can schedule \(M_s\) map tasks on \(S\) map worker nodes with SSDs and \((M - M_s)\) map tasks on \(H\) map worker nodes with HDDs. In this paper, \(M_s\) is calculated according to \([M \times \frac{\sum_j (S \text{ speed }_j)}{\sum_j (S \text{ speed }_j \times \text{Capacity }_j)}]\) because the I/O execution time (e.g., \(S \text{ speed}\)) is considered first and other factors (e.g., \(\text{Capacity}\) or \(\text{Lifetime}\)) can be considered afterwards.

After \(M_s\) is determined, \(M_s\) map tasks can be scheduled on \(S\) map worker nodes with SSDs. The situation is similar to Case I, as shown in the following:

- If \(M_s \leq S\), we can choose \(M_s\) worker nodes with SSDs according to the \(SCL\) weighting function (i.e., \(S \text{ speed } \times \text{Capacity } \times \text{Lifetime}\)). For \(i = 1\) to \(M_s\), \(t_i^{m_s}\) is dispatched to the map worker node with the largest \(SCL\) weighting value.
- If \(M_s > S\), we can schedule \(M_s\) map tasks on \(S\) map worker nodes according to the following scheduling rule: Assume that \(s_i\) denotes a map worker node, whose \(SCL\) weighting value is \(S \text{ speed }_i \times \text{Capacity }_j \times \text{Lifetime }_k\). When \(M_s\) map tasks are dispatched, \(s_i\) can get \([M_s \times \frac{S \text{ speed }_i \times \text{Capacity }_j \times \text{Lifetime }_k}{\sum_j (S \text{ speed }_j \times \text{Capacity }_j \times \text{Lifetime }_k)}]\) map tasks.

We can schedule \((M - M_s)\) map tasks on \(H\) map worker nodes according to the following scheduling rule. The situation is similar to Case II, as shown in the following:

- If \((M - M_s) \leq H\), we can choose \((M - M_s)\) map worker nodes with HDDs according to the \(SC\) weighting function (i.e., \(S \text{ speed } \times \text{Capacity}\)). For \(i = 1\) to \((M - M_s)\), \(t_i^{M_s-H}\) is dispatched to the map worker node with the largest \(SC\) weighting value.
- If \((M - M_s) > H\), we can schedule \((M - M_s)\) map tasks on \(H\) map worker nodes with HDDs according to the following scheduling rule: Assume that \(h_i\) denotes a map worker node, whose \(SC\) weighting value is \(S \text{ speed }_i \times \text{Capacity }_j\). When \((M - M_s)\) map tasks are dispatched, \(h_i\) map worker node can get \([(M - M_s) \times \frac{S \text{ speed }_i \times \text{Capacity }_j}{\sum_j (S \text{ speed }_j \times \text{Capacity }_j)}]\) map tasks.

4.3 Reduce Phase

When \(R\) is determined, the partition function is used for each map task to generate intermediate data into \(R\) regions in its local disk. After the intermediate data are partitioned into \(R\) regions (in their local disks) by all map tasks, \(R\) reduce worker nodes are required to perform \(R\) reduce tasks. When a reduce worker node is notified by the master node, it uses the remote procedure calls to collect its corresponding part of intermediate data from the local disk of each map worker node. Therefore, different reduce worker nodes could handle different data sizes due to the collection of the corresponding part from the \(R\) regions of each map worker node. Assume that \(R\) reduce tasks \((t_i^r \sim t_r^r)\) have been sorted in a ascending order according to the data size. We need to assign \(R\) reduce tasks to \((S + H)\) worker nodes. There are three cases that need to consider:

4.3.1 Case I: All \(R\) Reduce Tasks Are Performed by \(S\) Reduce Worker Nodes with SSDs

- If \(R \leq S\), we can choose \(R\) reduce worker nodes with SSDs according to the \(SCL\) weighting function (i.e., \(S \text{ speed } \times \text{Capacity } \times \text{Lifetime}\)). The dispatching rule is to dispatch each reduce tasks to the reduce worker node with the largest \(SCL\) weighting value. We list the pseudo code in the following Algorithm 3:
If $R > S$, we can schedule $R$ reduce tasks on $S$ reduce worker nodes with SSDs according to the following scheduling rule: For $i = 1$ to $R$, $t'_i$ is dispatched to the reduce worker node with the largest $SCL$ weighting value. Note that when a reduce worker node is chosen, it cannot be selected again until all $S$ reduce worker nodes have been selected.

**Algorithm 3** Case I for ($R \leq S$)

1. Let $SCL$ be $Speed \times Capacity \times Lifetime$
2. Calculate each reduce worker node’s $SCL$ weighting value
3. for $i = 1$ to $R$
4. $t'_i$ is dispatched to the reduce worker node with the largest $SCL$ weighting value
5. end for

4.3.2 Case II: All $R$ Reduce Tasks Are Performed by $H$ Reduce Worker Nodes with HDDs

- If $R \leq H$, we can choose $R$ reduce worker nodes with HDDs according to the $S$ weighting function (i.e., $S \times Capacity$). The dispatching rule is to dispatch each reduce task to the reduce worker node with the largest $SC$ weighting value. We list the pseudo code in the following Algorithm 4:

- If $R > H$, we can schedule $R$ reduce tasks on $H$ reduce worker nodes with HDDs according to the following scheduling rule: For $i = 1$ to $R$, $t'_i$ is dispatched to the reduce worker node with the largest $SCL$ weighting value. Note that when a reduce worker node is chosen, it cannot be selected again until all $H$ reduce worker nodes have been selected.

**Algorithm 4** Case II for ($R \leq H$)

1. Let $SC$ be $Speed \times Capacity$
2. Calculate each reduce worker node’s $SC$ weighting value
3. for $i = 1$ to $R$
4. $t'_i$ is dispatched to the reduce worker node with the largest $SC$ weighting value
5. end for

4.3.3 Case III: Part of $R$ Reduce Tasks Is Performed by $S$ Reduce Worker Nodes with SSDs and the Rest Is Performed by $H$ Reduce Worker Nodes with HDDs

In the case, our concern is to reduce the I/O execution time when the storage speed is considered. We consider the data size of reduce tasks and the storage speed because the size of each reduce task could be different and could affect the system performance. Therefore, we partition $R$ reduce tasks ($t'_1 \sim t'_R$) into a set of $R_s$ reduce tasks on $S$ reduce worker nodes with SSDs and a set of $(R - R_s)$ reduce tasks on $H$ reduce worker nodes with HDDs. We need to find a maximum $R_s$ such that

$$\frac{\text{total data size of } (t'_1 \sim t'_R)}{\text{total data size of } (t'_1 \sim t'_R)} \leq \frac{\sum_{i=1}^{R_s} \text{speed}_i}{\sum_{i=1}^{R_s} \text{speed}_i + \sum_{i=R_s+1}^{R} \text{speed}_i},$$

because we want to reduce the I/O execution time when the storage speed is considered. After the partition, the set of $(t'_1 \sim t'_R)$ is handled like Case I, and the set of $(t'_{R_s+1} \sim t'_R)$ is handled like Case II.

5. Performance Evaluation

5.1 Experimental Setup and Metrics

The goal of the experiments is to examine the performance of the hybrid storage system by taking into account the effect of SSDs and HDDs for the map-reduce framework in a virtualized environment. We used ten virtual machines for cluster nodes. The host machine was equipped with an Intel Core(TM) i7-4770 CPU@3.40GHz, 32GB RAM, 1TB HDD, 1TB SSD, and Windows 8 Enterprise operating system. We used ten virtual machines, with one master node and one user node. The master node is used to manage the overall execution of the map-reduce computation. The user node is used to provide the input data as a map-reduce job. The remaining eight virtual machines are worker nodes that serve as map worker nodes and reduce worker nodes. A map worker node is a type of worker node that can perform the map function and a reduce worker node is another type of worker node that can perform the reduce function. Each worker node in a virtual machine was equipped with 3GB RAM, Ubuntu 14.04 LTS operating system, and 56GB HDD or SSD based up on the case of experiments to be conducted. In order to have equal storage size for each virtual machine, the storage allocation (e.g., 56GB HDD and SSD) has been done by considering the free space of 1TB HDD and 1TB SSD. Because we propose a virtualization-based hybrid storage system for a map-reduce framework, we can consider the effects of the map-reduce framework in the virtualized environment. However, we still should consider the effects of sharing a physical HDD or SSD by virtual machines. Therefore, we suggest when the master node collect the required parameters of storage devices (e.g., $Speed$, $Capacity$, and $Lifet ime$) from all map worker nodes and reduce worker nodes, the amount of written data can also be counted for each physical SSD. Because $Capacity$ denotes the available free space of an SSD and $Lifet ime$ denotes the maximum allowable amount of data written to an SSD divided to the actual total amount of data written to the SSD, the master node can use $Capacity$ and $Lifet ime$ to count the amount of written data for each physical SSD.

In addition, the input split size and buffer size was fixed to 128MB. In the experiments, three different sizes of the input data included 5GB, 10GB, and 15GB. The maximum size of the input data must consider the storage allocation for each virtual machine and the intermediate data generated during the map-reduce computation. The size of the intermediate data depends on the number of the map tasks (i.e., the input splits) assigned to the map worker node or the
number of reduce tasks assigned to the reduce worker node. In fact, before the end of the map-reduce computation, the intermediate data could not be deleted to avoid the unnecessary performance degradation. Furthermore, we don’t delete the intermediate data in order to investigate the effects of the map-reduce computation on the storage characteristics (e.g., Speed, Capacity, or Lifetime). For example, if we delete the intermediate data, the storage capacity of each worker node could be the same and thus we can’t observe the effects of the map-reduce computation on the storage capacity.

We have developed our test platform using C/C++ based on the design [1]. Our test platform contains four nodes: master node, map worker node, reduce worker node, and user node. The master node will manage the overall scheduling and the execution of the map-reduce computation. The map worker node is to perform the map function and the reduce worker node is to perform the reduce function. Because we implement the design [1], the master node acts as a job tracker and the worker nodes act as task trackers, where a job tracker is the node to manage the overall execution of map-reduce jobs and a task tracker is the worker node to execute map or reduce tasks. When the user node executes a map-reduce application, the master node can collect the required parameters of storage devices (e.g., Speed, Capacity, and Lifetime) from all map worker nodes and reduce worker nodes. In fact, the information (e.g., Speed, Capacity, and Lifetime) is only required before a map computation or a reduce computation, and the information should be updated after a map computation or a reduce computation is finished. Therefore, the communization overhead on the master node is not significant in the experiments. With the required parameters of storage devices (e.g., Speed, Capacity, and Lifetime), the master node can calculate the weighting function (e.g., Speed * Capacity * Lifetime). Based on the calculated weighting value, the master node can partition the input data into the map tasks and assign/schedule the map tasks to the appropriate map worker nodes. After the map phases, the reduce tasks are produced and the master node also bases on the required parameters of storage devices (e.g., Speed, Capacity, and Lifetime) to assign/schedule the reduce tasks to the appropriate reduce worker nodes. The design idea behind the proposed method is that more appropriate worker nodes can handle more tasks to improve performance in terms of Speed, Capacity, and Lifetime.

In the experiments, we measured the execution time of the sorting computation in the test platform. In particular, sorting is one type of I/O-intensive workloads and involves a lot of I/O operations in both map and reduce phase of the map-reduce computation [14]. Because the objective of the proposed method is to combine the advantages of the fast access property of SSDs and the low cost of HDDs by realizing an economical design and improving I/O performance of a map-reduce framework in a virtualization environment, we determine the I/O-intensive workloads such as sorting in the experiments. In fact, the execution time of the sorting computation is determined by the slowest worker node to finish its task [12]. The measurement of the execution time includes mapping and partitioning time, shuffling time, and reducing time. Based on the proposed method, we measured the execution time for three cases as follows:

- Case I: Map and Reduce Worker Nodes Run on SSD-based Worker Nodes
- Case II: Map and Reduce Worker Nodes Run on HDD-based Worker Nodes
- Case III: Map and Reduce Worker Nodes Run on the Hybrid of SSD-based and HDD-based Worker Nodes

5.2 Experimental Results and Discussion

5.2.1 Case I: Map and Reduce Worker Nodes Run on SSD-Based Worker Nodes

In the experiment, we set up a map-reduce cluster that consists of one master node equipped with an HDD, four map worker nodes equipped with an SSD, and four reduce worker nodes equipped with an SSD. The number of map tasks (i.e. the input data size divided by the input split size) varies with the input data size, and the number of reduce tasks is fixed to six. We measured the execution time for three different sizes of the input data to test the performance of sorting computation. As shown in Fig. 2, the execution time increased when the input data size was large. Figure 3 also depicts the size of the total input splits assigned to a map worker node with the largest execution time for Case I.
Table 1 Number of input splits assigned to each map worker node for 15GB input data size — Case I

| Measurement                                      | Map Worker Node |
|-------------------------------------------------|-----------------|
| The Weighting Value (i.e., SCL) Divided by 1,000 | M0             | 14,656 |
| Number of Input Splits                          | M1             | 12,025 |
|                                                  | M2             | 14,321 |
|                                                  | M3             | 13     |

Table 2 Number of reduce tasks assigned to each reduce worker node for 15GB input data size — Case I

| Measurement                                      | Reduce Worker Node |
|-------------------------------------------------|--------------------|
| The Weighting Value (i.e., SCL) Divided by 1,000 | R0             | 14,218 |
| Number of Reduce Tasks                          | R1             | 14,264 |
|                                                  | R2             | 6,658  |
|                                                  | R3             | 7,590  |

Table 3 Number of input splits assigned to each map worker node for 15GB input data size — Case II

| Measurement                                      | Map Worker Node |
|-------------------------------------------------|-----------------|
| The Weighting Value (i.e., SC) Divided by 1,000  | M0             | 10,607 |
| Number of Input Splits                          | M1             | 6,658  |
|                                                  | M2             | 5,388  |
|                                                  | M3             | 7,590  |

Table 4 Number of reduce tasks assigned to each reduce worker node for 15GB input data size — Case II

| Measurement                                      | Reduce Worker Node |
|-------------------------------------------------|--------------------|
| The Weighting Value (i.e., SC) Divided by 1,000  | R0             | 8,906  |
| Number of Reduce Tasks                          | R1             | 6,798  |
|                                                  | R2             | 6,764  |
|                                                  | R3             | 7,582  |

of I/O operations.

Table 1 shows the number of the input splits assigned to each map worker node for 15GB input data size. As we can see from Table 1, the map worker node with the largest weighting value (i.e., the SCL weighting function) will be assigned more number of input splits because its storage system could have better speed, capacity, and lifetime. Similarly, in Table 2, we can see that the reduce worker node with the largest weighting value will be assigned more number of reduce tasks. Assigning more number of input splits and reduce tasks to the worker node with the largest weighting value can improve performance in the map-reduce framework.

5.2.2 Case II: Map and Reduce Worker Nodes Run on HDD-Based Worker Nodes

In the experiment, we set up a map-reduce cluster that consists of one master node equipped with an HDD, four map worker nodes equipped with an HDD, and four reduce worker nodes equipped with an HDD. The number of map tasks (i.e. the input data size divided by the input split size) varies with the input data size, and the number of reduce tasks is fixed to six. Similar to Case I, we measured the execution time for three different sizes of the input data to test the performance of sorting computation. As shown in Fig. 4, the execution time increased when the input data size was large.

When we compare Fig. 2 with Fig. 4, it shows that Case II performs better than Case I on 5GB and 10GB input data sizes. If we observe Fig. 3 and Fig. 5, the size of the total input splits assigned to a map worker node with the largest weighting value for Case II is smaller than Case I on 5GB and 10GB input data sizes. For example, for 5GB input data size, Case II assigned 1,408MB input splits to a map worker node with the largest weighting value, while Case I assigned 2,304MB input splits to a map worker node with the largest weighting value. This is because the test platform calculates the number of input splits using the formula \[ M = \frac{\sum \text{Speed} \times \text{Capacity}}{\sum \text{Speed} \times \text{Capacity} + \sum \text{Speed} \times \text{Capacity} \times \text{Lifetime}} \] for Case II and \[ M = \frac{\sum \text{Speed} \times \text{Capacity}}{\sum \text{Speed} \times \text{Capacity} + \sum \text{Speed} \times \text{Capacity} \times \text{Lifetime}} \] for Case I, which could generate different numbers of input splits for different values of capacity, speed, and lifetime. Therefore, the execution time of Case II was better than Case I for 5GB input data size. Table 3 and Table 4 show that more number of input splits and reduce tasks will be assigned to the more appropriate map and reduce worker nodes. This is because more tasks should be assigned to the worker node with the largest weighting value to improve performance.
5.2.3 Case III: Map and Reduce Worker Nodes Run on the Hybrid of SSD-Based and HDD-Based Worker Nodes

In the experiment, we set up three configurations of a map-reduce cluster based up on the number of SSD-based worker nodes and HDD-based worker nodes. The first configuration consists of one master node equipped with an HDD, one map worker node equipped with an SSD, three map worker nodes equipped with an HDD, one reduce worker node equipped with an SSD, and three reduce worker nodes equipped with an HDD. The number of map tasks (i.e. the input data size divided by the input split size) varies with the input data size, and the number of reduce tasks is fixed to six. The execution time of the configuration is shown in Fig. 6. The second configuration consists of one master node equipped with an HDD, two map worker nodes equipped with an SSD, two map worker nodes equipped with an HDD, two reduce worker nodes equipped with an SSD, and two reduce worker nodes equipped with an HDD. The number of map tasks (i.e. the input data size divided by the input split size) varies with the input data size, and the number of reduce tasks is fixed to six. The execution time of the configuration is shown in Fig. 7. The third configuration consists of one master node equipped with an HDD, three map worker nodes equipped with an SSD, one map worker node equipped with an HDD, three reduce worker nodes equipped with an SSD, and one reduce worker node equipped with an HDD. The number of map tasks (i.e. the input data size divided by the input split size) varies with the input data size, and the number of reduce tasks is fixed to six. The execution time of the configuration is shown in Fig. 8.

As shown in Fig. 6, Fig. 7, and Fig. 8 that are similar to Case I and Case II, the execution time increased when the input data size was large. When we compare the execution time of the third configuration (three SSD-based worker nodes and one HDD-based worker node) with that of the first configuration (one SSD-based worker node and three HDD-based worker nodes), we can see that the third configuration can perform well. This shows that the more the number of SSD-based worker nodes, the better the performance but more cost. The first configuration that consists of more HDDs doesn’t have good performance. Therefore, we suggest that we can adopt a small number of SSD-based worker nodes to seek a balance between economy and performance. For example, when we compare the third configuration with the second configuration (two SSD-based worker nodes and two HDD-based worker nodes), we can see that the third configuration performed better than the second configuration for 5GB and 10GB input data size, but the second configuration performed better than the third configuration for 15GB input data size. This is because the activities of garbage collection[13] inside the SSD could be triggered when the SSD-based worker nodes are used. Therefore, when the number of SSD-based worker nodes is increased and the input data is large, the performance degradation caused by the activities of garbage collection inside the SSD should be considered. In order to have an economical design and reasonable performance, we select the second configuration (that consists of equal number of SSD-based and HDD-based worker nodes) as Case III to measure the execution time. After selecting the second configuration as Case III, we have to conduct an experiment to compare the performance of three cases (Case I, Case II and Case III). We conduct the experiment on the same input data size (i.e 15GB) and on the same input split distribution. That is, the number of input splits assigned to each worker node should be uniform in all cases. According to the experimental results in Table 5, we can see that Case I performs better than Case II and Case III. Also we can see that Case III performs better than Case II by 2.36%. This shows that Case III can provide better performance and more economical cost than Case II.
Table 5  Comparison of Case I, Case II, and Case III performance on 15GB input data size

| Cases      | Map partition time | Shuffling time | Reducing time | Total time (sec) |
|------------|--------------------|----------------|---------------|------------------|
| Case I     | 14517.09           | 1089.42        | 1844.61       | 17451.12         |
| Case II    | 14489.46           | 945.88         | 2438.93       | 18784.27         |
| Case III   | 13878.4            | 878.67         | 2705.31       | 17462.38         |

Table 6  Number of input splits assigned to each map worker node for 15GB input data size: one SSD-based worker node and three HDD-based worker nodes

| Measurement                          | M0(HDD) | M1(HDD) | M2(HDD) | M3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 10500   | 10174   | 10635   | 12284   |
| Number of Input Splits               | 28      | 27      | 27      | 38      |

Table 7  Number of reduce tasks assigned to each reduce worker node for 15GB input data size: one SSD-based worker node and three HDD-based worker nodes

| Measurement                          | R0(HDD) | R1(HDD) | R2(HDD) | R3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 9763    | 9818    | 9654    | 11883   |
| Number of Reduce Tasks               | 1       | 1       | 1       | 3       |

Table 8  Number of input splits assigned to each map worker node for 15GB input data size: two SSD-based worker nodes and two HDD-based worker nodes

| Measurement                          | M0(HDD) | M1(HDD) | M2(SSD) | M3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 10732   | 10691   | 12386   | 12942   |
| Number of Input Splits               | 26      | 25      | 34      | 35      |

Table 9  Number of reduce tasks assigned to each reduce worker node for 15GB input data size: two SSD-based worker nodes and two HDD-based worker nodes

| Measurement                          | R0(HDD) | R1(HDD) | R2(SSD) | R3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 9611    | 9709    | 17356   | 17691   |
| Number of Reduce Tasks               | 1       | 1       | 2       | 2       |

Table 10  Number of input splits assigned to each map worker node for 15GB input data size: three SSD-based worker nodes and one HDD-based worker node

| Measurement                          | M0(HDD) | M1(SSD) | M2(SSD) | M3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 11183   | 12965   | 13182   | 13310   |
| Number of Input Splits               | 24      | 32      | 33      | 31      |

Table 11  Number of reduce tasks assigned to each reduce worker node for 15GB input data size: three SSD-based worker nodes and one HDD-based worker node

| Measurement                          | R0(HDD) | R1(SSD) | R2(SSD) | R3(SSD) |
|--------------------------------------|---------|---------|---------|---------|
| The Weighting Value (i.e., S/C/SCL) Divided by 1,000 | 9710    | 8293    | 7826    | 11442   |
| Number of Reduce Tasks               | 1       | 2       | 1       | 2       |

Overall, we think that the virtualization overhead (i.e., VMware) and the characteristics of SSDs (i.e., garbage collection) could have impact on the improvement ratio of Case I and Case III. For example, in Case I and Case III, the SSD space is partitioned to eight and four partitions to serve eight and four virtual machines (i.e., worker nodes), respectively. When the free SSD space is not enough, the activities of garbage collection [13] inside the SSD to recycle the invalid data could be triggered frequently and cause additional overhead for Case I and Case III. Therefore, we consider the virtualization overhead and the characteristics of SSDs by conducting experiments in the virtualization-based hybrid storage system. We also demonstrate that Case III could alleviate the impact of garbage collection inside the SSD because of its hybrid of SSD-based and HDD-based worker nodes. Note that Table 6, Table 7, Table 8, Table 9, Table 10, and Table 11 show that more number of input splits and reduce tasks will be assigned to the more appropriate map and reduce worker nodes under three configurations.

6. Conclusion

We propose a virtualization-based hybrid storage system for a map-reduce framework that combines the advantages of the fast access property of SSDs and the low cost of HDDs to improve I/O performance and cost efficiency in a virtualization environment. In the proposed method, the characteristics of HDDs (e.g., speed and capacity) and SSDs (e.g., speed, capacity, and lifetime) are considered to improve the map and reduce task execution time. We propose a general design that assumes that there are one master node and (S + H) worker nodes, where S worker nodes are equipped with SSDs and H worker nodes are equipped with HDDs. We propose how to assign M map tasks and R reduce tasks to (S + H) worker nodes balancing the characteristics of SSDs and HDDs. Therefore, we consider and discuss three cases that include SSD-based, HDD-based, and the hybrid of SSD-based and HDD-based worker nodes. Furthermore, we propose a weighting function to consider the characteristics of SSDs and HDDs (e.g., speed, capacity, and lifetime).
and use the weighting function to assign map and reduce tasks to the most appropriate map and reduce worker nodes. The experimental results show that the proposed method can improve performance by 2.36% when compared to HDD-based storage systems. In addition, because we use the hybrid of SSD-based and HDD-based storage systems, it is obviously more economical than SSD-based storage systems. As a result, the hybrid of SSD-based and HDD-based storage systems offers superior performance and economy.

For future work, we should further explore the performance and benefits of storage virtualization for the map-reduce framework. The storage virtualization can affect the performance of the map-reduce framework due to its internal activities such as data migration and data duplication. We think further investigation into the storage virtualization for the map-reduce framework is required and will become an important issue, especially for different applications with different resource requirements.

References

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” In OSDI, vol.51, no.1, pp.107–113, 2004.
[2] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee, “A case for flash memory SSD in Hadoop applications,” International Journal of Control and Automation, vol.6, no.1, Feb. 2013.
[3] M. Wasi-ur-Rahman, N.S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D.K.D. Panda, “High-performance RDMA-based design of Hadoop MapReduce over InfiniBand,” IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, Cambridge, MA, pp.1908–1917, 2013.
[4] K. Kambatla and Y. Chen, “The truth about MapReduce performance on SSDs,” 28th Large Installation System Administration Conference (LISA14), USENIX Association, Seattle, WA, 2014.
[5] S. Moon, J. Lee, Y.S. Kee, “Introducing SSDs to the Hadoop MapReduce Framework,” IEEE 7th International Conference on Cloud Computing (CLOUD), pp.272–279, June 27–July 2 2014.
[6] D. Jiang, B.C. Ooi, L. Shi, and S. Wu, “The performance of MapReduce: an in-depth study,” Proc. VLDB Endowment, 36th International Conference on Very Large Data Bases, vol.3, no.1, pp.472–483, Singapore, Sept. 13-17, 2010.
[7] http://en.wikipedia.org/wiki/Hard_disk_drive
[8] http://www.ptechguide.com/hard-disks/hard-disk-hard-drive-performance-transfer-rates-latency-and-seek-times
[9] http://en.wikipedia.org/wiki/Solid-state_drive
[10] S.S. Rizvi and T.-S. Chung, “Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems,” 2nd International Conference on Computer Engineering and Technology (ICCET), vol.7, Chengdu, pp.297–299, 16–18 April 2010.
[11] Santo Domingo, Joel (Feb. 18, 2015), “SSD vs HDD: What’s the Difference?”, PC Magazine Retrieved, Aug. 15, 2015.
[12] E. Bortnikov, A. Frank, E. Hillel, and S. Rao, “Predicting execution bottlenecks in map-reduce clusters,” Proc. 4th USENIX Conference on Hot Topics in Cloud Computing, USENIX Association, Berkeley, CA, USA, 2012.
[13] R. Subramani, H. Swapnil, N. Thakur, B. Radhakrishnan, and K. Puttaiha, “Garbage collection algorithms for NAND flash memory devices – an overview,” European Modelling Symposium (EMS), Manchester, USA, pp.81–86, 20-22 Nov. 2013.
[14] J. Issa and A. Kassem, “Disk I/O performance-per-Watt analysis for cloud computing,” International Journal of Computer Applications, vol.97, no.3, pp.23–27, July 2014.
[15] S. Lee, T. Kim, K. Kim, and J. Kim, “Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling,” Proc. 10th USENIX on File and Storage Technologies, p.26, Berkeley, CA, USA, 2012.
[16] C.-K. Kang, Y.-J. Cai, C.-H. Wu, and P.-C. Hsu, “A Hybrid Storage Access Framework for High-Performance Virtual Machines,” ACM Transactions on Embedded Computing Systems (ACM TECS), vol.13, no.5s, pp.1–24, 2014.
[17] Y.-J. Cai, C.-K. Kang, and C.-H. Wu, “A Virtual Storage Environment for SSDs and HDDs in Xen Hypervisor,” ACM SIGBED Review (Special Interest Group on Embedded Systems), vol.11, no.2, pp.39–44, June 2014.
[18] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, “BlueDBM: An Appliance for Big Data Analytics,” International Symposium on Computer Architecture (ISCA), Portland, OR USA, June 13-17, 2015.
[19] F. Chen, D.A. Koufaty, and X. Zhang, “Hystor: making the best usage of Solid State Drives in high performance storage systems,” Proc. ACM Int’l Conf. on Supercomputing (ICS 2011), pp.22–32, 2011.
[20] 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,” Proc. USENIX on Networked Systems Design and Implementation, 2012.
[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Petterly, “Dryad: distributed data-parallel programs from sequential building blocks,” Proc. ACM SIGOPS/EuroSys European Conference on Computer Systems, pp.59–72, 2007.
[22] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, “RAID: High-Performance, ReliableSecondary Storage,” ACM Computing Surveys, vol.26, no.2, pp.145–185, June 1994.
[23] S. Gupta and M. Rogers, “Exploring Forensic Implications of the Fusion Drive,” JDFS L V9N2, 2014.
[24] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, “Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications” Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD ‘15), ACM, New York, NY, USA, pp.1357–1369, 2015.

Aseffa Dereje Tekilu received his B.S. degree in Electrical Engineering from Mekelle University in 2008, Mekelle, Ethiopia. He received his M.Tech. degree in Electronics and Computer Engineering from Addis Ababa University in 2010, Addis Ababa, Ethiopia. He is currently working toward the Ph.D. degree in the Electronic and Computer Engineering from National Taiwan University of Science and Technology, Taipei, Taiwan. He is also a member of IEEE. His research interests include Cloud Computing and Big Data Analytics.

Chin-Hsien Wu received his B.S. degree in computer science from National Chung-Cheng University in 1999. He received his M.S. and Ph.D. degree in computer science from National Taiwan University in 2001 and 2006, respectively. Now, he is an associate professor at the department of Electronic and Computer Engineering in National Taiwan University of Science and Technology. He is also a member of ACM and IEEE. His research interests include embedded systems, real-time systems, and flash-memory storage systems.