Attributed Consistent Hashing for Heterogeneous Storage Systems

Jiang Zhou*†, Yong Chen*, Weiping Wang†
*Texas Tech University, Lubbock, Texas, USA
†Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

ABSTRACT

Storage systems are critical building blocks of high-end computing systems and data centers. They demand the flexibility to distribute data effectively and provide high I/O performance. The consistent hashing algorithm is widely used in parallel/distributed file systems due to its decentralized design, scalability, and adaptability to node changes. However, it lacks efficiency in a heterogeneous environment where different storage devices, e.g. hard disk drives and solid state drives, co-exist. In this study, we propose an *attributed consistent hashing (attributedCH)*, to overcome this deficiency. AttributedCH manages heterogeneous nodes on a consistent hashing ring and maintains attributes for each node to characterize distinct node features. It divides the hash ring into sectors and selects nodes from the sector with a comprehensive selection strategy. By considering different attributes, attributedCH achieves adaptive and efficient data placement for heterogeneous storage systems. We have carried out extensive evaluations and the evaluation results confirm that the attributedCH overcomes the deficiency of existing consistent hashing algorithms well and is particularly suitable for heterogeneous storage systems.

CCS CONCEPTS

• **Information systems → Information storage systems**; *Distributed storage*; • **Computer systems organization →** *Heterogeneous (hybrid) systems*;

KEYWORDS

Parallel file system, data distribution, heterogeneous storage system, consistent hashing

ACM Reference Format:
Jiang Zhou*†, Yong Chen*, Weiping Wang†. 2018. Attributed Consistent Hashing for, Heterogeneous Storage Systems. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT'18).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3243176.3243202

1 INTRODUCTION

Metadata management has become a critical challenge that large-scale data centers and high-end computing systems face nowadays. Traditional parallel/distributed file systems support a global, hierarchical namespace for files contained in storage clusters. They require centralized or distributed metadata servers to handle the mapping between data objects/items and storage nodes. The metadata server essentially uses a mapping table to manage $< data\ item, node >$ distribution [1, 2]. However, as the scale of the system grows, the size of the management table becomes enormously large. The metadata server becomes a performance bottleneck for data mapping in the storage system. The synchronization of such a large table between different metadata servers becomes time-consuming too. Additionally, the mapping table based design potentially limits the system's scalability because it is on the path of every data access.

On the other hand, the hashing-based data management approaches eliminate the need of metadata servers for mapping data to nodes. They use hashing algorithms, such as consistent hashing [3] or pseudo-random number functions [4–6], to determine the distribution of data on storage nodes. The benefits of them, compared to the mapping table, include decent performance, high scalability, easy management, and adaptation to node changes without completely reorganizing data layout. Numerous existing storage systems with hashing-based distribution have gained increasing attention, such as the Dynamo [7], Cassandra [8], Ceph [9], Sheepdog [10], and GlusterFS [11].

The conventional consistent hashing algorithm assigns a set of items to "buckets" so that each bucket receives roughly the same number of items, resulting balanced storage usage [3]. This design ensures balanced data distribution, assuming each storage node has the same specification. A *heterogeneous* storage system contains various classes of storage devices, and these existing consistent hashing algorithms will not work well (a consistent hashing variant with virtual nodes is discussed below). For instance, given devices with very different capacity and bandwidth, it is a trade-off to place data by considering the capacity usage and bandwidth usage, instead of only considering data distribution based on capacity. Heterogeneous (or hybrid) storage systems can be a promising trend since they leverage the benefits of both cost-effective hard disk drives and fast-but-expensive new types of storage devices such as solid state drives (SSDs), phase change memory (PCM) [12], and other storage class memory (SCM) devices. Heterogeneous storage systems are more likely to be the norm given the deep storage hierarchy in data centers and high-end computing systems.

Numerous data distribution algorithms have been proposed for heterogeneous storage systems. Most hashing-based approaches focus on considering the capacity difference among different storage nodes. For instance, CRUSH [4, 9], consistent hashing with using virtual nodes [3, 10, 13], SPOCA [14], and ASURA [15] take storage capacity into consideration to distribute data proportionally. These strategies address the problem of "capacity" balance on storage nodes, but such a capacity-based distribution ignores the performance difference (e.g. the bandwidth) of storage nodes, which is a sub-optional design. Apparently SCM devices have much smaller capacity than hard disk drives, but they provide significantly higher bandwidth [12], which raises the problem that the capacity-based distribution can under-utilize fast SCM devices because of their lower capacity. What is even worse, when the I/O access is intensive, hard disk drives can be overwhelmed because they are assigned a large portion of I/O load. Several existing studies have also explored the performance improvement by leveraging fast devices [16–24]. They distribute data in heterogeneous storage systems with optimized layout for the entire system's I/O efficiency. However, most of them mainly consider the performance benefits of storage systems, not synthetical features such as capacity utilization and data availability. Some approaches [25–27] consider capacity, performance, and other factors for data placement, but they are not designed for hashing-based storage systems.

In this paper, we present a novel consistent hashing-based data distribution approach called *attributed consistent hashing (attributedCH)*, which places data in a heterogeneous environment effectively. To the best of our knowledge, attributedCH is the first study to introduce an *attribute* concept into data distribution hashing algorithms. More specifically, attributedCH manages heterogeneous nodes on a hash ring, and maintains attributes like capacity, bandwidth, and location, for each node on the ring. These attributes characterize distinct features of nodes and are then used for data distribution. When placing data, attributedCH divides the hash ring into sectors, and selects nodes from the sector via a selection strategy. It keeps the inherent property of consistent hashing, with only a small amount of data movement when node membership changes. By considering different node attributes, attributedCH strikes a balance among capacity utilization, performance (bandwidth), and availability. We have carried out extensive evaluations, and the evaluation results confirm that attributedCH makes effective use of heterogeneous devices and addresses the limitation of existing consistent hashing algorithms well.

2 BACKGROUND AND MOTIVATION

Consistent Hashing. Although various data distribution algorithms are designed for parallel/distributed storage systems, consistent hashing is widely used for decentralized node management, high scalability, and adaptability to node changes [3]. It uses hash functions to map data items/objects onto nodes. In addition, only a small amount of data objects will migrate for distribution balance when node addition (e.g., scale out the storage system) or removal (e.g., node fails) occurs.

Traditional consistent hashing uses a *hash ring*, a hypothetical data structure that contains a list of hash values that wraps around at both ends. It first hashes the ID number of storage nodes and sorts their hash values to form the hash ring. Then, given a key (i.e. the object ID), it hashes the key to a position on the hash ring, and continues walking along the ring in a clockwise direction from the position until finding the first node/position (also called a successor of the hashed key), and returns the associated node for storing/retrieving the data object. When there are r replicas, consistent hashing will choose r closest nodes along the ring because redundant copies are usually used in distributed storage systems. Figure 1(a) illustrates an example of data distribution in consistent hashing with replica number (also called replication factor) $r = 3$. Data x is hashed to a position on the hash ring, and then walking clockwise from that position, the next node (a successor) on the ring is selected for data placement or retrieval, i.e. node A in this example. By continuing to walk along the hash ring, the second replica ($r2$) and third replica ($r3$) are placed on node B and C.

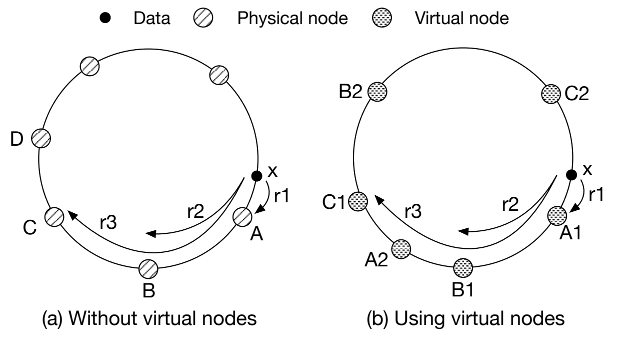


Figure 1: Data distribution with traditional consistent hashing, given the replication factor $r = 3$

Consistent Hashing with Virtual Nodes. To achieve balanced data distribution, consistent hashing uses virtual nodes (VNodes) to distribute data more uniformly. Each physical node may have multiple virtual nodes, which are responsible for multiple positions assigned along the hash ring. If a virtual node is encountered, the physical storage node associated with that virtual node is selected to place the data object. Figure 1(b) shows an example of consistent hashing using virtual nodes, where each physical node has two virtual nodes. For instance, $A1$ and $A2$ are virtual nodes of the same physical node A. Data x is hashed to the hash ring and selects its next virtual node $A1$ for data placement. Note that consistent hashing will skip a virtual node if there is already a copy of the data on the associated physical node for data availability. Thus, the second replica $r2$ is placed on $B1$, but the third replica $r3$ skips $A2$ and is placed on $C1$ because there is already one replica on $A1$.

Virtual nodes can be used to adjust data placement for heterogeneous environments [3, 10, 11, 28]. By assigning a different number of virtual nodes, consistent hashing can distribute data among different nodes proportionally according to their *storage capacity*. For instance, an HDD node with $1TB$ can have four times of the virtual node number of an SSD node with $256GB$ for capacity-based data distribution.

The data can also be replicated on different physical nodes to improve data redundancy and availability. For instance, HekaFS [13] tunes the consistent hashing with multi-ring hashing. Instead of each node having multiple tokens (virtual nodes) on one ring, it has one token in multiple rings. HekaFS uses part of the hashing value to select a ring, then the remaining value is used to map data in that ring. It is mathematically equivalent to assign multiple tokens in a singe ring, while each token of different nodes composes a separate ring. Multi-ring hashing ensures that replicas are placed on physical nodes in a non-overlapping way, but cannot distinguish heterogeneous node features.

Limitations of Existing Approaches. The existing consistent hashing and alike algorithms (both with and without virtual nodes), however, have a critical limitation for data distribution in *heterogeneous* storage systems, i.e. they only consider the capacity, not any other features of heterogeneous nodes, such as bandwidth and location (i.e the rack of nodes). Since different devices have distinct features, e.g. SCM devices offer much higher bandwidth but with significantly smaller capacity compared with HDDs, current consistent hashing-based distribution will under-utilize fast SCM devices. In other words, such a "balanced" distribution based on capacity only is essentially "imbalanced" given different features like bandwidth, location of devices in heterogeneous storage systems. The designs of existing approaches are sub-optional for heterogeneous storage systems.

Given the limitations of existing approaches in heterogeneous storage systems, we introduce a new *attributed consistent hashing (attributedCH)* data placement approach that can take different node attributes into consideration to be well suited for heterogeneous storage systems.

3 ATTRIBUTED CONSISTENT HASHING

The goal of the proposed attributed consistent hashing (attributedCH) data placement approach is to distribute data among *heterogeneous* nodes effectively by well considering their distinct characteristics. To distinguish different node characteristics, attributedCH introduces an *attribute* concept into consistent hashing for the first time. With this new concept, each node on the hash ring has associated *attributes*, and these attributes will be considered in data placement decision. Our current design primarily focuses on three attributes: *capacity*, *bandwidth*, and *location*, with considering capacity utilization, I/O performance, and data availability. It provides an adaptive data distribution on heterogeneous environments while keeping the inherent property of consistent hashing. We discuss details below.

3.1 Node attributes

In a heterogeneous system equipped with HDDs, SSDs, and other storage devices, nodes may have different characteristics. In order to distinguish them, we introduce an attribute concept in consistent hashing, and more specifically, we focus on three attributes in the current study. Other attributes can also be added to extend our design.

The first attribute we consider is *capacity*. We select the capacity as an attribute because the proportional distribution of data according to the capacity is important for heterogeneous storage systems. If the data is not distributed well, it will result in load imbalance where nodes with smaller capacity can be overloaded, while other nodes with larger capacity can have plenty of free space.

The second attribute we consider is *bandwidth*. We select the bandwidth as an attribute because there is a significant gap among different storage devices, which concerns the I/O performance of the storage system. Faster devices with higher throughput will reduce the data-access time for applications.

The third attribute we consider is node location. In a large-scale storage system, nodes are connected with different topologies, such as the flat topology with rack-node structure [2] or a hierarchical cluster map [9]. Placing data on various nodes and racks can improve the I/O concurrency for read and write operations. It can also improve the data availability, as long as replicas are distributed at different locations. For example, when a node is powered down, the data on that node can still be retrieved from its replica on another node or rack. Since the node location is an important factor for data accesses, we choose it as one node attribute for data placement.

Admittedly, other attributes can also influence data placement. AttributedCH is designed with focusing on these three most important attributes, and can also be extended for considering other attributes.

3.2 Hash ring with attributes

Like consistent hashing, attributedCH uses a single hash ring to assign and manage nodes on the ring. The main difference is that attributedCH adds attributes for each node on the hash ring and considers these attributes for data placement.

Suppose there is a set of n nodes $V = \{v_1, \ldots, v_n\}$. We use a tuple $< c_i, b_i, l_i >$ to describe the attributes of node v_i, where c_i denotes the node capacity, b_i denotes the node bandwidth, and l_i denotes the node location. All these attributes are predefined. For capacity, we use the total capacity as the attribute c_i for each node. For bandwidth, we use the average bandwidth as the attribute b_i. For the location, we use the "zone" as the attribute l_i and to distinguish nodes in different racks [10]. This means that if two nodes are in different racks, they have different values of zone. There are two reasons why we choose these predefined values rather than dynamic ones. First, a dynamic value of an attribute may disturb the data distribution, which makes the data position change dynamically. It is not feasible to move data frequently in a large-scale storage system. Second, it will complicate the design if we need to monitor or synchronize the node status for defining attributes.

To illustrate with an example, assume we have a storage system with three heterogeneous storage nodes $\{v_1, v_2, v_3\}$ with different attributes $< 1024, 146, zone0 >$, $< 512, 196, zone0 >$, and $< 256, 540, zone1 >$, respectively. The attribute $< 1024, 146, zone0 >$ means that node v_1 is in rack $zone0$ with the capacity $1024GB$ (or 1TB) and bandwidth $146MB/s$. The attributes of two other nodes are similarly defined. From the location attribute, it can be seen that v_1 and v_2 are in the same rack, while v_3 is in a different rack.

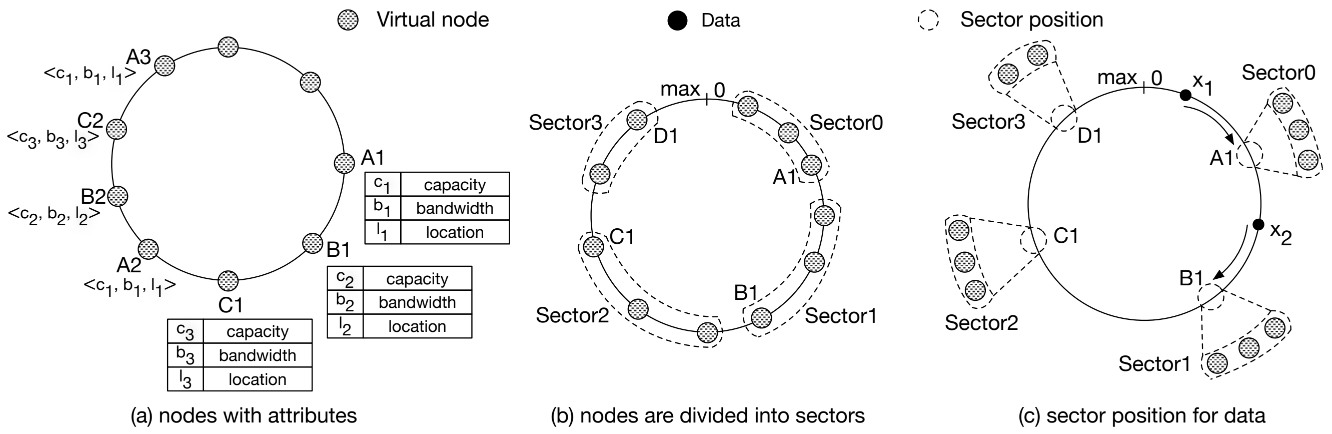


Figure 2: Consistent hash ring in attributedCH. Each node has three attributes with virtual nodes. Nodes are divided into sectors for data placement.

Virtual nodes are also supported in the attributedCH. To generate virtual nodes, the naive approach is to use $\left\lceil \frac{c_i}{min_{capacity}} \right\rceil$ virtual nodes for each node v_i, where $min_{capacity}$ is the minimum capacity of all nodes [28]. However, more nodes will be involved in data movement for node changes if $\left\lceil \frac{c_i}{min_{capacity}} \right\rceil$ is too large [29]. It can also require a large memory footprint and communication cost [30]. To address this issue, we use a throttling value $c_{throttle}$ to generate virtual nodes. We set the default value of $c_{throttle}$ to $1GB$. The number of virtual nodes assigned to node v_i on the hash ring is calculated with $\left\lceil \frac{c_i}{c_{throttle}} \right\rceil$. Then, each node has a different number of virtual nodes according to the capacity attribute. To make a trade-off between virtual node management and data movement, we use $65,536$ as the upper bound for the virtual node number [29].

Since each virtual node is a copy of its associated physical node, we assign the same attributes to the virtual node as its physical node. Figure 2(a) shows the hash ring with different node attributes, where a few nodes are illustrated on the ring. It can be seen that each node has three attributes and a number of different virtual nodes. Specifically, nodes $A1$, $A2$, and $A3$ are virtual nodes of the physical node A, and they have the same attributes as node A.

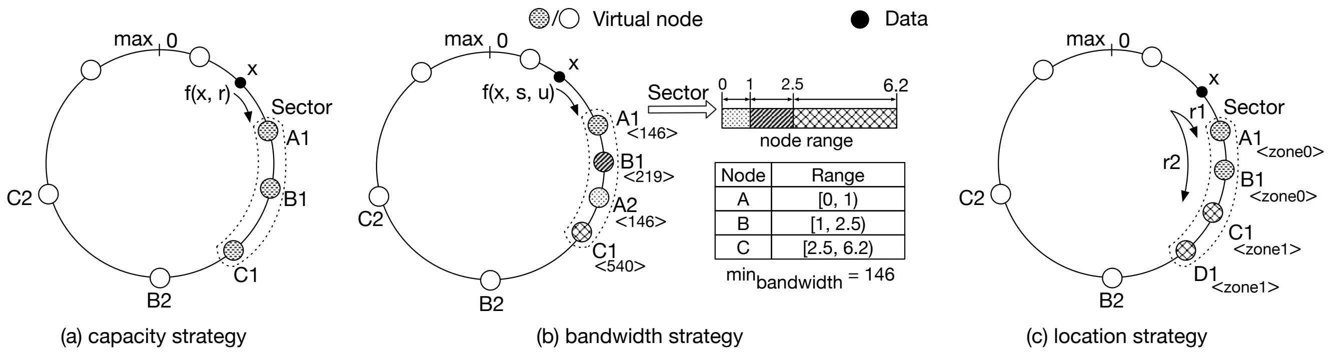
3.3 Sectors in the hash ring

Traditional consistent hashing uses hash functions to assign data to nodes on a hash ring. Suppose that nodes are mapped into the hash ring using a hash function $h_1 : V \rightarrow [0, max)$, where max is the upper bound of hash value on the ring. Then, the data items $X = \{x_1, \ldots, x_m\}$ are mapped on the same ring using a hash function $h_2 : X \rightarrow [0, max)$. Consistent hashing achieves an even data distribution, but it cannot distinguish different node characteristics, because the data position is randomly selected by the hash values. In contrast to this, attributedCH uses node attributes on the hash ring for data placement. To distinguish between different node attributes, attributedCH selects a group of nodes as candidate nodes on the hash ring. It chooses an appropriate node to store data with a selection strategy that uses attributes in the group. By considering different node attributes, attributedCH can well consider merits of heterogeneous nodes with respect to capacity utilization, I/O performance, and data availability.

To enable this idea, attributedCH first divides nodes on the hash ring into sectors. Each sector is consisted of a group of nodes, in which the sector size $s (s < n)$ is defined in advance (but can change when a node is added or removed, discussed in detail in Section 3.5). To divide sectors, attributedCH starts from the position 0 on the ring and selects s successor nodes clockwise as the first sector. The next s successor nodes after the first sector are assigned into the next sector. Using this method, attributedCH divides nodes into different sectors until it reaches the max value on the ring. The last sector may have fewer nodes than s. Suppose there are n' virtual nodes on the hash ring. Then, there are total $\left\lceil \frac{n'}{s} \right\rceil$ sectors.

Note that the sector size will not affect the data distribution, as attributedCH can select appropriate node inside each sector (see evaluations in Section 4.2.1). Figure 2(b) shows an example of hash ring with sectors. It can be seen that all nodes on the ring are divided into four sectors. Specifically, nodes $A1$, $B1$ and $C1$ belong to three sectors with each sector size 3. Node $D1$ is in the last sector, which contains 2 nodes. The sectors do not take the location of nodes (or zones) into account, but attributedCH considers the location as a single attribute when placing replicas. There is no need to assure that sectors are uniform in terms of capacity, bandwidth, and location, or contain the same number of physical nodes. The reasons are two-fold. First, attributedCH is exactly proposed to address issues of conventional consistent hashing, i.e. only focuses on balanced storage utilization, not in terms of multiple factors. Second, a comprehensive selection strategy will be used in the sector and will be discussed in detail in Section 3.4.

By dividing the hash ring into sectors, attributedCH maps data onto nodes in two steps. First, it finds the sector on the ring with the hash value of the data. Next, it uses a selection strategy to choose appropriate nodes in a sector according to node attributes. Note that if we treat all nodes on the ring as one sector, we can only distribute data by considering one factor of node attributes



Node	Range
A	[0, 1)
B	[1, 2.5)
C	[2.5, 6.2)

Figure 3: Three strategies for data placement in attributedCH: the *capacity strategy* handles the capacity attribute for space utilization goal; the *bandwidth strategy* focuses on the bandwidth attribute for performance goal; the *location strategy* uses location attribute to improve data availability.

(capacity or bandwidth, discussed in detail in Section 3.4). In the first step, attributedCH will find a sector for the data. In order to maintain sectors on the ring, we use the *last node* in a sector to represent the sector's position. By comparing hash values, the data will be assigned to its first successor sector. Figure 2(c) shows the search of sector position for data on the hash ring where the sectors are the same as in Figure 2(b). Working in the clockwise direction, node $A1$ is the last node in $sector0$ (with sector ID 0). Thus, all data whose hash values are between 0 and the position of node $A1$ (which is also the position of $sector0$) will be assigned to $sector0$. Similarly, data x_2 is assigned to $sector1$ because its first successor sector is $sector1$, whose position is node $B1$.

3.4 Consistent hashing strategy

Generally speaking, attributedCH is designed to reconcile two competing objectives: distributing data on heterogeneous nodes efficiently and taking full advantage of their characteristics. To achieve the objectives, we divide the hash ring into sectors to distribute data with node attributes, instead of just using hash values. In a heterogeneous environment, a sector may consist of HDD nodes, SCM nodes, or both (which may be homogeneous or heterogeneous). To distinguish different nodes, we use a selection strategy in attributedCH for data placement according to capacity, bandwidth, and location, depending on the node attributes in a sector.

3.4.1 Capacity strategy. If a sector consists of homogeneous nodes, such as HDD or SSD nodes, attributedCH uses a capacity strategy for data placement. This strategy takes the capacity attribute into account with a uniform data distribution. It is natural to evenly distribute data on nodes/virtual nodes of the sector in two cases. The first case is that the HDD or SSD nodes have similar attributes. There is no significant bandwidth gap among homogeneous nodes, which causes them to be treated equally. The other case is that we have made different virtual nodes for each physical node, with the storage capacity. The data can be proportionally distributed to nodes on the ring according to their capacity.

Given an input value of data x, the data is first mapped to a position on the ring with a hash value $hash(x)$, where the hash function can be the *MD5* function or the *FNV-1* function [10]. As discussed above, attributedCH finds the first successor sector for the data, and then selects nodes in the sector as candidates. If all nodes in a sector are homogeneous, attributedCH uses a capacity strategy for data placement. In this strategy, a node is chosen from the sector with the expression $f(x) = (hash(x) + p)\ mod\ s$, where p is a randomly (but deterministically) chosen prime number greater than s [4]. If there are r replicas, attributedCH picks the rest of placement nodes following the expression $f(x, r) = (hash(x) + rp)\ mod\ s$, where r is the r^{th} replica for the data x. Figure 3(a) shows an example of data placement with the capacity strategy, where data x is first assigned to its successor sector, and then to the nodes in that sector.

3.4.2 Bandwidth strategy. In a heterogeneous environment, a sector is often consisted of hybrid nodes, such as HDD and SSD nodes. These nodes have significant performance gap in I/O accesses. To distinguish them, we use a bandwidth strategy for data placement. In this strategy, attributedCH places data among nodes in a sector using a pseudo-random number algorithm. It places data among nodes proportionally with respect to their bandwidth.

We assign a range for each node in a sector according to its node bandwidth. The range of all nodes in the sector forms a number line, where each range is ordered with the start value, 0. A sector may include multiple virtual nodes for the same physical node. As virtual nodes are generated using node capacity, we only use one virtual node to represent the bandwidth of any given physical node. We assign one range for a physical node and ignore other virtual nodes because this strategy focuses on the bandwidth attribute. Otherwise, for instance, an HDD node can be assigned twice of data than an SSD node, which is against the bandwidth strategy goal (a HDD node has lower bandwidth than that of an SSD node). The range length is calculated via dividing the node bandwidth by the minimum bandwidth $min_{bandwidth}$ in the sector:

$$range\ length = \frac{node\ bandwidth}{min_{bandwidth}}$$

Unlike the node position on the hash ring, the node range is only used within the sector. Each sector has separate nodes with different ranges. For data placement, attributedCH first selects the sector with the data hash value $hash(x)$. If the sector consists of HDD and SSD nodes, the data is assigned to one range (which represents one node) by the pseudo-random hash function. A random number in a given range $[0,\ u)$ is generated for data mapping. The computation of a random number val can be described with the expression below, where x is data ID, s is a seed, and u is the upper bound of the random number (0 is always a constant lower bound). The hash function f can be a pseudo-random number generator or SIMD-oriented Fast Mersenne Twister (SFMT) algorithm [31].

$$val \Leftarrow f(x,s,u)$$

Figure 3(b) shows an example of data placement in one sector using the bandwidth strategy. We only illustrate the bandwidth attribute for each node. It can be seen that there are four virtual nodes in the sector. Given the $min_{bandwidth} = 146$, these four virtual nodes are assigned to three ranges of different lengths in a number line. Note that as node A has two virtual nodes, $A1$ and $A2$, it is only assigned to a range $[0,1)$ for the virtual node $A1$. The virtual node $A2$ is ignored without assigning a range. The length of each range indicates the bandwidth attribute of a node. After constructing the range number line, attributedCH uses hash functions to select the node position. For example, the data x will be placed on node $B1$ if the random number generated for the data is in the range $[1,2.5)$. In this manner, the node with higher bandwidth will have greater chance to be selected to store data. If data is replicated with multiple copies, multiple pseudo-random numbers will be generated. If the seed for a data x is the same, the same random number will be generated, which makes the data distribution and look up deterministic.

3.4.3 *Location strategy.*

The capacity and bandwidth strategies place data on nodes according to their capacity or bandwidth attributes. To better utilize node characteristics, we use the location strategy to improve data availability.

Although our capacity and bandwidth strategies can select appropriate nodes for individual data items, the replicas of one data may be placed in the same location, such as the same node or rack. To address this issue, attributedCH checks node location for replicas before node selection. AttributedCH tries to select a node for the r^{th} replica that has a different location from the $r-1$ other replicas. It will skip the current node (or zone) and search the next node (or zone) in a sector, until it finds a different, valid node location, or reaches the last node of a sector. This can improve data availability, because the data can be retrieved from another node or zone if a node, or even an entire rack, crashes. Besides, one sector may include multiple virtual nodes for the same physical node. To reduce the probability of node collision, attributedCH will also skip a virtual node if any of the data exists on its corresponding physical node. Thus, our strategy replicates data by distributing replicas on different node locations.

Figure 3(c) shows an example of data placement in one sector with the location strategy. In the figure, we only give the location attribute for each node. Suppose two replicas of data x, $r1$ and $r2$, should map to node $A1$ and node $B1$ with hashing algorithms. However, node $B1$ has the same location ($zone0$) as node $A1$. Therefore, the *second* replica, $r2$, will skip node $B1$ and will be placed on node $C1$, which is the next node with a different location from the node $A1$ of replica $r1$.

3.4.4 *Strategy selection.*

We have introduced three strategies for data placement in heterogeneous environments. Each strategy can be used independently to achieve data distribution. For instance, Sheepdog [10] and GlusterFS [11] use virtual nodes to adjust data placement, like using the capacity strategy. HDFS [2] and HekaFS [13] distribute data on different nodes, which can be considered as applying the location strategy. However, using one type of strategy will only provide a sub-optimal data placement for heterogeneous nodes because each strategy only considers one attribute of nodes.

Algorithm 1 Data placement and replication algorithm

INPUT: data ID x, replica number r, sector size s;

```
1:  procedure DATA PLACEMENT($x$, $r$, $s$)
2:      ▷initialize hash ring with sectors
3:      $ch\_initialize(s)$
4:      ▷select r successor sectors of data x
5:      $sects[] \Leftarrow ch\_find\_closest(x,\ r)$
6:      for $i = 0;\ i < r;\ i++$ do
7:          if $sects[i]$ is homogeneous then
8:              $node = select\_node(capacity\ strategy)$
9:          else
10:             $node = select\_node(bandwidth\ strategy)$
11:         end if
12:         while node has replica of data x do
13:             $check\_node(location\ strategy)$
14:             $node = select\_node(capacity\ strategy,$
15:                         $or\ bandwidth\ strategy)$
16:         end while
17:         ▷assign x to selected node in the sector
18:         $assign\_node(sects[i],\ x)$
19:     end for
20: end procedure
```

In order to take full advantage of heterogeneous node features, attributedCH combines these three strategies together and uses a comprehensive strategy to select appropriate nodes for data placement. If a data item is mapped to a homogeneous sector, attributedCH will use the *capacity strategy*; otherwise, it will use the *bandwidth strategy*. When making r replicas for the data x, attributedCH selects r successor sectors of the data. It places the first replica with the *capacity strategy* or *bandwidth strategy* in a sector. For each one of remaining $r-1$ replicas, attributedCH first hashes the data to a node and then applies the *location strategy* to check whether the node already has the copy of data. If the node has the data replica, attributedCH will skip it and find the next node in a different location with *capacity strategy* or *bandwidth strategy*, depending on the sector setting. Algorithm 1 describes the data replication and placement algorithm with the strategy selection in attributedCH.

To further explain the process, Figure 4 shows an example of strategy selection in attributedCH. Given the replica number of

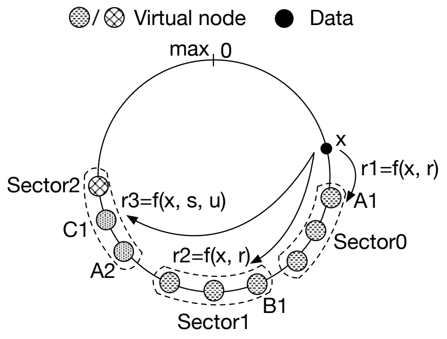


Figure 4: Strategy selection, $A1$ and $A2$ are virtual nodes of the same physical node A

3, the data x is mapped to its three successor sectors. Suppose $sector0$ and $sector1$ are homogeneous sectors, and $sector2$ contains heterogeneous nodes. For the first replica, attributedCH use the capacity strategy to place data on $A1$ in $sector0$, specifically as determined by the hashing value of $f(x,r)$. For the second replica, data x is first hashed to $B1$ in $sector1$, also with the capacity strategy. At this time, attributedCH uses the location strategy to check node location. As $A1$ and $B1$ are in different locations, attributedCH places the second replica on $B1$. For the third replica, attributedCH uses the bandwidth strategy to hash data in $sector2$. Suppose the data is first hashed to $A2$, it is hashed again to $C1$ because the location strategy finds out that there is already one copy in node A ($A1$ in $sector0$).

3.5 Coping with node changes

In a large-scale storage system, the cluster scale may change by adding or removing nodes. Consistent hashing provides an advantage for data distribution, because the mapping from data to nodes is perturbed as little as possible when the number of nodes is changed. The only data that needs to move is relatively a small amount of data whose node assignment changed. Since the attributedCH is based on consistent hashing, it is important to maintain the inherent property of consistent hashing for data movement when node changes occur.

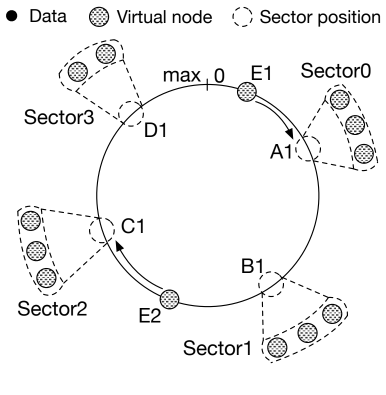


Figure 5: Add a new node on the consistent hashing ring

For node addition, attributedCH maps the new node on the hash ring and assigns the node to its first successor sector. If the node has multiple virtual nodes, each virtual node will be assigned to a sector according to its position. A sector may also contain more than one virtual node. Other sectors that are not the neighbors of the new node will not change their sizes. This means that attributedCH merges the new node into existing sectors, without creating a new sector for it. The benefit of this is that, when moving data, attributedCH only needs to migrate data among the nodes in the sector that contains the new node. As the new node has its own attributes, the selection strategy for data placement in the sector that has the new node may be changed. For instance, when an SSD node is added into a sector with homogeneous HDD nodes, the strategy will change from *capacity strategy* to *bandwidth strategy*. Otherwise, the positions of remaining data are recalculated with the hash function $(hash(x) + rp)\ mod\ s'$, where s' is the new size of the sector, or with $f(x,s,u')$, where u' is the new upper bound of range in the sector. Figure 5 shows an example of adding a new node on the hash ring, where sectors are the same as in Figure 2(b). It can be seen that node E has two virtual nodes, where $E1$ and $E2$ are merged into $sector0$ and $sector2$, respectively.

For node removal, attributedCH deletes the node along with its virtual nodes from all sectors on the hash ring. Data movement may only occur in the sector where nodes are removed. When there is a node removal in a sector, we do not decrease the size of the sector but simply *mark* the removed node and *skip* it for data placement. This process may increase the load on the next node, but such is the nature of a consistent hashing data placement algorithm. For a sector with homogeneous nodes, the remaining data will not move for node deletion, because we can still use the same hash function with the capacity strategy. For a sector with heterogeneous nodes, the selection strategy may change from *bandwidth strategy* to *capacity strategy* for node removal. In this case, the remaining data will move by recalculating the new position with the hash function used with the *capacity strategy*. Otherwise, the remaining data do not need to move, because the hash values calculated by the *bandwidth strategy* will remain the same.

When nodes are added/removed, the number of nodes per sector may be non-uniform, but a sector's position will not change. Thus, we can find the sector on the ring with the hash value of the data, and choose an appropriate node. Compared with conventional hashing functions (e.g. round-robin assignment), the data position recalculation overhead for node changes in attributedCH incurs with two factors: a small amount of data movement and computation time for calculating the new position, which is $O(logn)$ and acceptable. Detailed evaluations are presented in Section 4.1.2.

4 EXPERIMENTAL EVALUATION

In this section, we present experimental evaluation results of our approach, attributedCH. We have evaluated it from two aspects: first, we conducted a simulation test in a modular consistent hashing library named *libch-placement* [29]. The *libch-placement* library implements the consistent hashing algorithm, and is capable of simulating flexible settings with different hash functions, distance

Table 1: The calculation time of different algorithms (s)

Node No.	Consistent hashing	CRUSH (straw buckets)	attributedCH
64	0.0124	1.4591	0.0131
512	0.0204	9.1039	0.0218
4096	0.0332	70.2185	0.0361
32768	0.1783	582.4153	0.1948

metrics and virtual nodes. Second, we implemented the attributedCH approach based on Sheepdog [10], a distributed object-based storage in data centers for virtual machine (VM) storage. We used the FNV-1 hash function [10] for consistent hashing and the capacity strategy of attributedCH, and the SIMD-oriented Fast Mersenne Twister (SFMT) [31] algorithm to generate pseudo-random numbers for data placement in the bandwidth strategy of attributedCH.

The experiments were conducted on a local storage cluster with 30 nodes. Of these nodes, half of them are equipped with dual 2.6 GHz Xeon 8-core processors, 32GB memory, and a 200GB Intel SSD (SSDSC2BA200G3T); others have dual 2.5 GHz Xeon 8-core processors, 32GB memory and a 500GB Seagate SATA HDD (ST9500620NS). In actual tests, the average bandwidth of SSD is nearly 2.5 times than that of HDD. Each storage node can be used as a client node for the running of virtual machines (VMs). A VM instance was emulated by KVM-QEMU and configured with 2 vCPUs and 4GB RAM.

4.1 Simulation test for data placement approaches

4.1.1 Calculation time and memory usage. In order to examine the calculation and memory usage, we conducted emulation tests on one node, based on the *libch-placement* library. We compare attributedCH with two typical data placement algorithms: consistent hashing and CRUSH (straw buckets) algorithms. For these tests, we used 256 virtual nodes for consistent hashing and initial $c_{throttle} = 1GB$ and sector size $s = 4$ for attributedCH. In attributedCH, we assume there are two types of nodes with attributes of $< 4096, 146, zone0 >$ and $< 1024, 540, zone1 >$, respectively. The nodes on CRUSH are configured in one bucket, and all the replica numbers are set to 3. We generated $100,000$ data IDs, and distribute them with different data placement approaches. The tests were run 5 times and the average results are reported.

Table 1 shows the calculation time of different algorithms. As expected, the time of data placement for consistent hashing follows the $O(logn)$ asymptotic trend, where n is the number of virtual nodes. This is because the processing is essentially a binary search in the sorted node array. The time of data placement of straw buckets grows linearly as the number of nodes increases, as it will compare the hash values of all nodes to place data. AttributedCH is based on consistent hashing to use an additional hash function for the strategy to select the placement node. As the value of hash function is determinate with the $O(1)$ time complexity, the overhead caused by strategy selection is negligible. Thus, the complexity of finding a node with hashing functions in attributedCH depends on the time to find a sector (generally $O(logn)$, more specifically $O(log(n/sector\ size)))$ plus $O(1)$ to find a node in a sector. AtttributedCH scales well and the calculation time is very small as reported in Table 1.

Table 2: Memory footprint in different configurations

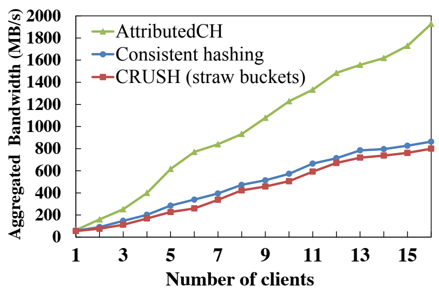
Node No. × VNode No.	Consistent hashing (MB)	attributedCH (MB)
100×256	0.72	0.98
500×256	3.51	5.18
1000×256	7.42	9.77
5000×256	39.06	54.83
10000×256	76.13	107.65
1000×1024	31.25	45.06
1000×4096	156.24	195.30
10000×4096	1562.4	1953.0

Memory usage is important for a data placement algorithm to be scalable. For consistent hashing, the data structure for all the virtual nodes needs to be stored in memory to allow fast calculation of the data locations (following the $O(32 \times Node\ No. \times VNode\ No.)$ space complexity). Compared with it, attributedCH adds three attributes to each virtual node's data structure and sector information (be maintained with *two* bytes on each virtual node structure: one is sector ID, the other is a flag indicating whether the sector contains homogeneous or heterogeneous nodes), increasing the size of each virtual node by more than ten bytes. To evaluate the memory usage and the feasibility of the data placement algorithm, we list the memory usage of the hashing algorithms with different configurations of $c_{throttle}$ for the number of nodes and number of virtual nodes for each node, as seen in Table 2.

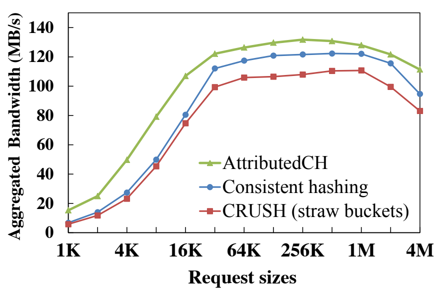
With 256 virtual nodes per physical node, the memory usage of consistent hashing and attributedCH range from $0.72MB$ to $107.65MB$ for 100 to $10,000$ storage nodes, which is acceptable considering that a few to tens GBs of main memory being the norm. The memory usage only becomes a problem when the number of nodes and virtual nodes are both large. For instance, a configuration with 10,000 nodes each with 4096 virtual nodes would require around $1.5GB$ and $1.9GB$ of memory for consistent hashing and attributedCH, respectively. The memory footprint can reach even higher if more virtual nodes are used. Indeed, our attributedCH can address this issue by using an appropriate number of virtual nodes according to heterogeneous node capacity. For instance, attributedCH can reduce the number of virtual nodes to achieve a high degree of fairness if the virtual node management occupies large amount of memory.

4.1.2 Evaluation of data movement. In order to evaluate the influence on data movement when node changes, we tested attributedCH for node addition with *libch-placement*. We assume the node configuration is the same as described in previous test. For attributedCH, we generated 4096 virtual nodes for HDDs, and 1024 for SSDs. We used 1024 as the virtual node number in consistent hashing.

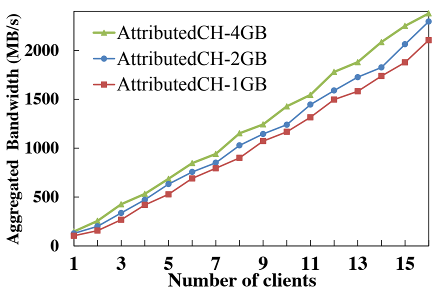
Figure 7 shows the number of data items that will be moved from each existing node to a newly added node. The x-axis indicates the



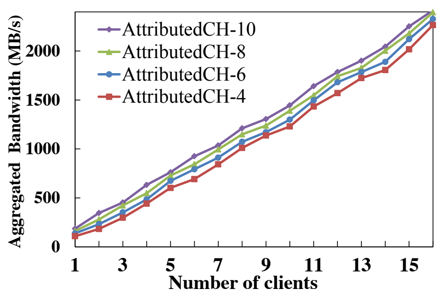
(a) Random read with different clients



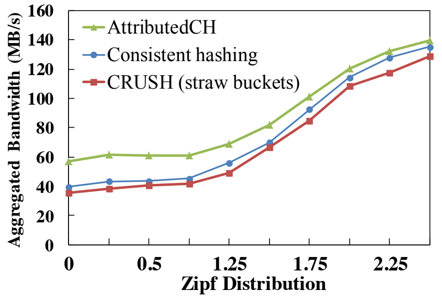
(b) Random read with different request sizes



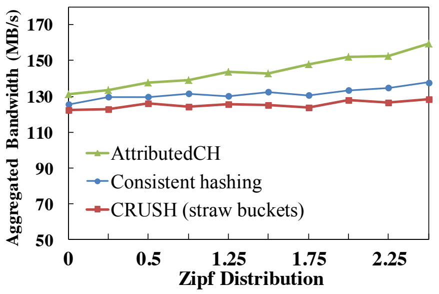
(c) Random read with different virtual nodes



(d) Random read with different sector sizes



(e) Ramdom I/Os with increasing skewness



(f) Sequential I/Os with increasing skewness

Figure 6: FIO performance comparison of three algorithms

node ID, indexed from 0, when the nodes are sorted by the amount of data movement. It can be seen that attributedCH follows the similar trend in data movement as consistent hashing. This is due to the fact that in attributedCH, data movement only occurs in the sector that contains newly added/removed nodes. The data layout in other sectors will not be affected. This evaluation shows that attributedCH can keep the inherent property of consistent hashing, without incurring a considerable amount of data movement overhead.

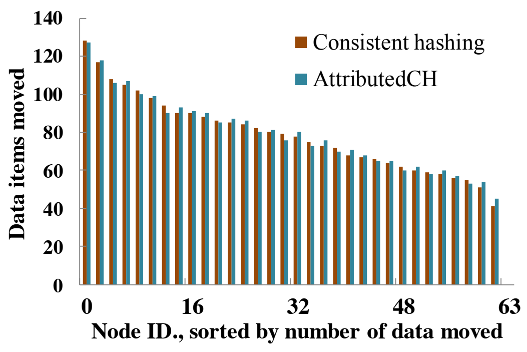


Figure 7: Data movement for a single node addition

4.2 Evaluation on distributed storage systems

We also evaluated attributedCH by comparing it with a baseline system based on Sheepdog [10], which provides block level storage volumes that can be attached to QEMU/KVM virtual machines. Sheepdog divides a virtual machine disk image (VMDI) into multiple fixed-size objects, and distributes them on storage nodes. It uses a consistent hashing-based algorithm for object placement, just as described in Figure 1. We implemented the attributedCH approach by modifying Sheepdog's data distribution and replication strategy. As Sheepdog provides an original consistent hashing algorithm, we use the unmodified Sheepdog system as a baseline. We also used a simple implementation of CRUSH algorithm [4], straw buckets, to compare its performance with our approach. Initially, the storage nodes are configured with 3 racks in which each consists of equal numbers of HDDs and SSDs. AttributedCH uses $c_{throttle} = 1GB$ as the parameter to allocate virtual nodes, and set the sector size $s = 4$. The Sheepdog storage cluster was formatted with a virtual node number of 128, and a default replica number of 3.

4.2.1 I/O performance. In this experiment, we emulated the read performance of data access with the FIO benchmark [32]. We use one or multiple clients running on different virtual machines to perform read operations, and obtain the aggregate bandwidth. Figures 6(a)-(d) show the random read performance with FIO for various algorithms. In Figure 6(a), we run from 1 to 16 clients which launch FIO on different VMs. Each FIO instance has 16 jobs, in which each job accesses an independent file with $100MB$ in an asynchronous way, with the request size of $256KB$. It can be observed that the performance of all algorithms improves with the increase of client number. This is because Sheepdog distributes objects on various storage nodes, and provides concurrent data access for clients. Compared with the consistent hashing and CRUSH algorithms, attributedCH shows a bandwidth improvement of 112% to 303%. Since attributedCH can use performance strategy to select nodes according to node attributes, it made full use of performance merits from SSD nodes (the observed ratio of heterogeneous and homogeneous sectors is 4 : 1). AttributedCH also achieved efficient capacity utilization with capacity strategy. Note that even though consistent hashing in Sheepdog makes 3 replicas for each object, the benefits are limited, as it cannot distinguish the HDD and SSD nodes [10]. For straw buckets, data are placed using the hash values of different nodes, also without considering node attributes.

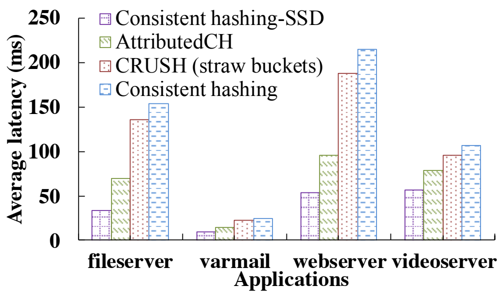
Figure 6(b) shows the results when we run FIO on one client, with request size ranging from 1KB to 4MB. When the request was smaller, the aggregate bandwidth became lower, as the storage node needs to serve more requests, which results in more time taken for disk seeking. Note that the average bandwidth will reach a peak value during tests. This is because the FIO performance in the VM is limited by the physical network bandwidth in the node the client was running. Similarly, the performance of attributedCH was 105% to 273% higher than that of consistent hashing and CRUSH algorithms. These tests confirmed the effectiveness and scalability of our data placement approach for I/O performance.

Figure 6(c)-(d) show the performance of attributedCH using different virtual nodes and sector sizes. These evaluations were conducted with multiple clients. The notations *attributedCH-2GB* or *attributedCH-4* mean that the parameter $c_{throttle}$ for making virtual nodes is *2GB* or the sector size is 4, respectively. In Figure 6(d), we only show the result for sector sizes ranging from 4 to 10, but the outcome is similar when increasing to a larger sector size. It can be seen that there is no significant performance effect when choosing different numbers of virtual nodes and sector sizes. This is because attributedCH can use an appropriate strategy for node selection, no matter how many virtual nodes are in the sector. For all data, attributedCH considers node attributes, and distributes the replicas in nodes with different locations. Similar results can be observed for the sequential I/O workload with FIO.

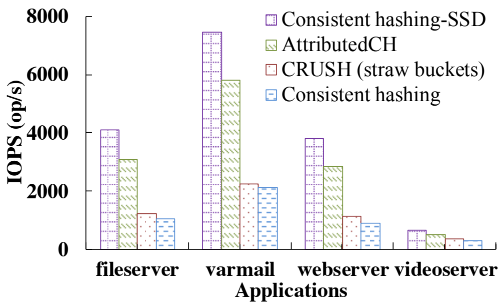
4.2.2 *Efficiency on skew data distribution.*

To further measure the performance impact by data placement approaches, we use imbalanced data access as the workload for evaluation. We ran the FIO benchmark with both random and sequential I/O operations, with increasingly skewed access distribution (Zipf distribution) [33]. The tests were conducted on Sheepdog with one client running on a virtual machine. The data distribution is more skewed with the increase of Zipf distribution value, ensuring that some part of the data are more frequently accessed than others.

Figure 6(e)-(f) show the read performance by comparing three algorithms. The results show that attributedCH achieved higher performance in both random and sequential access that other two algorithms. This is because attributedCH can select the *bandwidth strategy* according to node performance, and outperform SSDs. When the data are frequently accessed, more benefits are obtained from the SSD nodes. On contrary, the consistent hashing and CRUSH algorithms treat heterogeneous nodes uniformly and place data among them without considering node characteristics. With the Zipf distribution value being larger, the performance improvement by attributedCH becomes more significant.



(a) Average latency



(b) IOPS

Figure 8: Performance under different workloads

4.2.3 *File system workloads evaluation.*

In this subsection, we present the results of experiments conducted using the benchmark Filebench [34]. It emulates file system level workloads of real-world applications with different data access patterns. The workload specification is described in Table 3, and the *WF* indicates the operation of reading or writing a whole file. For comparison, we also tested the I/O performance by using all SSD nodes as storage nodes, namely consistent hashing-SSD. We launched one client running on a virtual machine on Sheepdog to get the experimental results.

Table 3: Specification of emulated file system workloads of different applications

Application	Dataset	R/W	Ave File Size	I/O size
fileserver	64GB	1:2	256KB	WF
varmail	32GB	1:1	32KB	WF
webserver	48GB	10:1	256KB	WF
videoserver	128GB	1:0	1GB	1MB

Figure 8 shows the evaluation results of different algorithms under fileserver, mailserver, webserver and videoserver workloads. It can be observed that among three algorithms, attributedCH offered the best performance, which is close to that of consistent hashing-SSD. The reason is that attributedCH can selectively place data on nodes according to their bandwidth. Data access on the

SSD nodes will significantly improve the overall I/O performance. On the other hand, the data are proportionally placed on nodes according to their capacity by virtual nodes in order to achieve data balance. The node location is also considered for data availability. In contrast, the consistent hashing and CRUSH algorithms place data among nodes evenly, without distinguishing different device merits. The results further confirm the efficiency of our attributedCH approach in a heterogeneous environment.

5 RELATED WORK

Numerous studies have been conducted in recent years regarding data distribution approaches and heterogeneous storage systems. We discuss existing work in this section, and compare them with this study.

5.1 Data placement approaches

Data placement approaches are widely used in many storage systems [1, 9, 10]. Most of them place data with deterministic hash mapping functions. GFS [35] and HDFS [2] divide data sets into fixed-size blocks, and distribute them with rack-aware data placement. Consistent hashing-based algorithms [3, 4] and storage systems [7, 8, 10, 36, 37], provide a scalable and uniform approach to place data on large-scale nodes. CRUSH [4] is a scalable pseudo-random data distribution function designed for a hierarchical cluster. It can map data objects to storage devices efficiently, while each bucket/hierarchy is essentially a homogeneous device.

There are also many methods based on data distribution/replication for storage performance optimization in parallel/distributed file systems. Song et al. [38] proposed a hybrid replication scheme for complex applications. Yin et al. [39] described an I/O data replication scheme to reorganize data according to the data access pattern. Similar strategies are designed to place data with hybrid redundant data distribution [40].

Different from the above studies, our proposed attributedCH introduces an attribute concept into data placement hashing algorithms. It manages different nodes on a consistent hashing ring, and maintains three attributes for each node on the ring. By considering node capacity, bandwidth and location, attributedCH selects appropriate nodes for data placement. With these attributes which characterize distinct node features, attributedCH achieves an adaptive data placement for heterogeneous storage, while keeping the inherent property of consistent hashing.

5.2 Heterogeneous storage systems

The heterogeneous storage architecture has become increasingly popular for massive data storage in high performance computing systems and data centers. Most existing data placement algorithms can efficiently and fairly distribute data in homogeneous storage, but do not meet the requirements of a hybrid environment [1–3]. Some algorithms address one device attribute (e.g., capacity) for data placement in a hybrid environment while overlooking other device characteristics [3, 4, 14, 15].

There is a rich body of related studies on heterogeneous storage systems with the consideration of different devices [18, 20, 21, 23, 25, 41]. Some of them combine data-access pattern analysis to achieve optimized data layout [16, 17, 19]. HiCH [42] introduces a hierarchical consistent hashing algorithm that divides HDD and SSD devices into two separate consistent hashing rings so that the SSD hash ring is regarded as the cache for the HDD hash ring. However, these studies lack an effective method to explore various and different device attributes (e.g., capacity, bandwidth, and location) in a heterogeneous storage cluster containing HDD, SSD and other SCM devices. SUORA [26] distributes data across a hybrid storage cluster by dividing different devices into buckets with considering their performance and assign data to devices in buckets based on their capacity. OctopusFS [27] employs multi-objective optimization techniques for making intelligent data management decisions in a hybrid environment. These solutions, however, are not designed for consistent hashing based storage systems, which we address in this research study.

6 CONCLUSION

The consistent hashing algorithm has been widely used in parallel/distributed storage systems due to its simplicity and scalability. It uses hash functions to map both the data and storage nodes to a hash ring, and then places data onto those storage nodes. However, current consistent hashing algorithms do not work well in a heterogeneous storage system with the co-existence of a variety of devices, as they do not distinguish different characteristics of heterogeneous devices.

This paper has introduced a novel consistent hashing based data placement approach, namely attributedCH (attributed consistent hashing), for heterogeneous storage systems. To the best of our knowledge, the attributedCH is a new consistent hashing method for heterogeneous storage that introduces an attribute concept for the first time. We have also introduced the attributedCH design details with considering capacity, bandwidth, and location attributes specifically. The attributedCH manages heterogeneous nodes on a consistent hash ring, and maintains attributes holistically in order to characterize distinct node features. For data placement, attributedCH divides the hash ring into sectors and uses a selection strategy to select data positions from a group of nodes in each sector. Data replication is also achieved to ensure data availability and reliability. We evaluated the attributedCH approach with both simulation tests and real-world benchmarks with a prototype implementation in Sheepdog storage system. The experimental results show that attributedCH overcomes the deficiency of existing consistent hashing algorithms well and is particularly suitable for heterogeneous storage systems.

ACKNOWLEDGMENT

This research is supported in part by the National Science Foundation under grant CNS-1338078, CNS-1526055, IIP-1362134, CCF-1409946, and CCF-1718336. This research is also supported by the Beijing Municipal Science and Technology Project under grant Z181100002718004.

REFERENCES

[1] P. J. Braam, "The Lustre storage architecture," White Paper, Cluster File System, Inc., Oct. 2003.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. of MSST'10*, Incline Village, USA, May 2010, pp. 1–10.

[3] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 654–663.

[4] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. of SC'06*, 2006.

[5] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[6] J. Lamping and E. Veach, "A fast, minimal memory, consistent hash algorithm," in *arXiv preprint arXiv:1406.2294*, 2014.

[7] G. DeCandia and et al., "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, 2007.

[8] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of OSDI*, 2006.

[10] "The Sheepdog Project Software," 2017. [Online]. Available: http://www.sheepdog-project.org/.

[11] A. Davies and A. Orsaria, "Scale out with GlusterFS," *Linux Journal*, 2013.

[12] H. Wong, S. Raoux, S. Kim, J. Liang, J. Reifenberg, B. Rajendran, M. Asheghi, and K. Goodson, "Phase change memory," *Proceedings of the IEEE*, 2010.

[13] J. Darcy, "HekaFS: Multi ring hashing," 2017. [Online]. Available: http://pl.atyp.us/hekafs.org/index.php/2012/07/multi-ring-hashing/

[14] A. Chawla, B. Reed, K. Juhnke, and G. Syed, "Semantics of caching with SPOCA: A stateless, proportional, optimally-consistent addressing algorithm," in *Proc. of the 2011 USENIX ATC*, 2011.

[15] K. I. Ishikawa, "ASURA: Scalable and uniform data distribution algorithm for storage clusters," *arXiv preprint arXiv:1309.7720*, 2013.

[16] S. He, X. Sun, and A. Haider, "HAS: Heterogeneity-aware selective layout scheme for parallel file systems on hybrid servers," in *Proc. of IPDPS'15*.

[17] S. Ma, H. Chen, Y. Shen, H. Lu, B. Wei, and P. He, "Providing hybrid block storage for virtual machines using object-based storage," in *Proc. of the ICPADS'14*.

[18] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Proc. of MSST'13*, 2013, pp. 1–12.

[19] F. Chen, D. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. of SC'11*, 2011, pp. 22–32.

[20] S. He, X. Sun, Y. Wang, and et al., "A heterogeneity-aware region-level data layout for hybrid parallel file systems," in *Proc. of the ICPP'15*, 2015.

[21] Z. Feng, Z. Feng, X. Wang, G. Rao, Y. Wei, and Z. Li, "HDStore: An SSD/HDD hybrid distributed storage scheme for large-scale data," *Web-Age Information Management*, pp. 209–220, 2014.

[22] J. Zhou, W. Xie, and Y. Chen, "Attributed consistent hashing for heterogeneous storage system," in *Proc. of the SC'16 (poster)*, 2016.

[23] L. Wu, Q. Zhuge, E. Sha, X. Chen, and et al., "Boss: An efficient data distribution strategy for object storage systems with hybrid devices," *IEEE Access*, 2017.

[24] H. Kim, D. Shin, Y. Jeong, and K. Kim, "SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device," in *the FAST'17*, 2017.

[25] W. Xie, J. Zhou, M. Reyes, J. Nobel, and Y. Chen, "Two-mode data distribution scheme for heterogeneous storage in data centers," in *Proceedings of The 2015 IEEE International Conference on Big Data*, 2015, pp. 327–332.

[26] J. Zhou, W. Xie, J. Noble, M. Reyes, and Y. Chen, "SUORA: A scalable and uniform data distribution algorithm for heterogeneous storage systems," in *Proc. of the NAS'16*, 2016.

[27] E. Kakoulli and H. Herodotou, "OctopusFS: A distributed file system with tiered storage management," in *Proc. of the SIGMOD'17*, 2017.

[28] C. Schindelhauer and G. Schomaker, "Weighted distributed hash tables," in *Proc. of the 17th SPAA*, 2005, pp. 218–227.

[29] P. Carns, K. Harms, J. Jenkins, M. Mubarak, R. B. Ross, and et al., "Consistent hashing distance metrics for large-scale object storage," in *Proc. of the SC'15, Poster.*

[30] P. Carns, K. Harms, J. Jenkins, M. Mubarak, and et al., "Impact of data placement on resilience in large-scale object storage systems," in *Proc. of the MSST'16*, 2016.

[31] "Simd-oriented fast mersenne twister," 2017. [Online]. Available: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/

[32] "The FIO Tool Benchmark." [Online]. Available: http://freecode.com/projects/fio.

[33] "Zipf Distribution," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Zipf's_law.

[34] "The File system benchmark," 2017. [Online]. Available: http://sourceforge.net/projects/filebench.

[35] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. of SOSP'03*, New York, USA, Oct. 2003, pp. 29–43.

[36] I. Stoica, R. Morris, D. Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *ACM SIGCOMM Computer Communication Review*, 2001.

[37] N. Zhao, J. Wan, J. Wang, and C. Xie, "GreenCHT: A power-proportional replication scheme for consistent hashing based key value storage systems," in *Proc. of MSST'15*, 2015.

[38] H. Song, Y. Yin, Y. Chen, and X. Sun, "A cost-intelligent application-specific data layout scheme for parallel file systems," in *Proc. of HPDC'11.*

[39] Y. Yin, J. Li, J. He, X. Sun, and R. Thakur, "Pattern-direct and layout-aware replication scheme for parallel I/O systems," in *Proceedings of the IPDPS'13.*

[40] B. Mao, S. Wu, and H. Jiang, "Improving storage availability in cloud-of-clouds with hybrid redundant data distribution," in *Proc. of IPDPS'15*, 2015, pp. 633–642.

[41] R. Dathathri, C. Reddy, T. Ramashekar, and U. Bondhugula, "Generating efficient data movement code for heterogeneous architectures with distributed-memory," in *Proc. of PACT'13*, 2013, pp. 375–386.

[42] J. Zhou, W. Xie, Q. Gu, and Y. Chen, "Hierarchical consistent hashing for heterogeneous object-based storage," in *Proc. of ISPA'16*, 2016.