CROCUS: Enabling Computing Resource Orchestration for Inline Cluster-Wide Deduplication on Scalable Storage Systems

Prince Hamandawana, Awais Khan, *Member, IEEE*, Chang-Gyu Lee, Sungyong Park, and Youngjae Kim, *Member, IEEE*

Abstract—Inline deduplication dramatically improves storage space utilization. However, it degrades I/O throughput due to compute-intensive deduplication operations such as chunking, fingerprinting or hashing of chunk content, and redundant lookup I/Os over the network in the I/O path. In particular, the fingerprint or hash generation of content contributes largely to the degraded I/O throughput and is computationally expensive. In this article, we propose CROCUS, a framework that enables compute resource orchestration to enhance cluster-wide deduplication performance. In particular, CROCUS takes into account all compute resources such as local and remote *{CPU, GPU}* by managing decentralized compute pools. An opportunistic Load-Aware Fingerprint Scheduler (LAFS), distributes and offloads compute-intensive deduplication operations in a load-aware fashion to compute pools. CROCUS is highly generic and can be adopted in both inline and offline deduplication with different storage tier configurations. We implemented CROCUS in Ceph scale-out storage system. Our extensive evaluation shows that CROCUS reduces the fingerprinting overhead by 86 percent with 4KB chunk size compared to Ceph with baseline deduplication while maintaining high disk-space savings. Our proposed LAFS scheduler, when tested in different internal and external contention scenarios also showed 54 percent improvement over a fixed or static scheduling approach.

Index Terms—Distributed file systems, scheduling, storage management

1 INTRODUCTION

THE explosive increase in data has made the problems of data storage space worse in the cloud [1], [2]. Cloud providers employ shared-nothing scale-out storage systems (SN-SS) such as Ceph [3] and GlusterFS [4] for high performance, scalability, and availability and to seamlessly suffice storage demands. In recent years, data deduplication [5], [6], [7], [8], a capacity optimization technique that is being used to dramatically improve storage efficiency, has emerged as an effective alternative to compression in backup storage for reducing storage demands. With the emergence of high-speed SSDs, the trend towards deploying tiered hybrid architecture is gaining more importance [9]. Configuring such hybrid systems (SSD and HDD) to achieve performance goals at a minimum cost remains a challenge [10]. Data deduplication on the SSDs would be at the forefront of the tiered architectures to minimize storage costs and increase SSD endurance without impairing storage performance.

- *P. Hamandawana is with the Department of Computer Engineering, Ajou University, Suwon 16499, Republic of Korea. E-mail: phamandawana@ajou.ac.kr.*
- *A. Khan, C. Lee, S. Park, and Y. Kim are with the Department of Computer Science and Engineering, Sogang University, Seoul 04107, Republic of Korea. E-mail: {awais, changgyu, parksy, youkim}@sogang.ac.kr.*

Manuscript received 28 July 2019; revised 28 Jan. 2020; accepted 31 Jan. 2020. Date of publication 11 Feb. 2020; date of current version 18 Mar. 2020. (Corresponding author: Youngjae Kim.) Recommended for acceptance by D. Talia. Digital Object Identifier no. 10.1109/TPDS.2020.2972882

Deduplication can be applied to the SSD tier either inline or offline. Inline deduplication introduces additional latency and performance degradation in the I/O path, whereas, offline deduplication requires high storage capacity in order to stage data temporarily. In particular, additional latency and performance degradation in inline deduplication are mainly attributed to compute-intensive operations of deduplication. This includes dividing an object into small fixed or variable-size chunks, computing fingerprint (FP) against each chunk using hash functions such as SHA1 or SHA512 and fingerprint indexing. Recently, there have been studies that have implemented cluster-wide inline deduplication in Ceph [11], [12]. These studies implemented cluster-wide deduplication following shared-nothing constraints, without impairing the scalability and fault tolerance of scale-out storage systems [11], [12]. However, in both studies, fingerprinting is computationally intensive, and in the case of small chunks, the performance degradation becomes critical due to frequent fingerprint operations.

Due to the many-core architectures in modern servers, there exist opportunities to overcome such latency and performance degradation problems in the current state of the art deduplication systems [13], [14]. By leveraging such powerful architectures, we can utilize the multi-threaded invocation of fingerprint operations to enhance the deduplication performance. On the other end, the CPU usage trend of the scale-out storage system suffocates the multi-thread deduplication performance [13]. For example in Ceph, the

frequent Object Storage Daemon (OSD) operations and internal cluster traffic such as replication, heartbeat messages, and storage rebalancing generates high CPU load, giving little room for performance improvement from the multi-threaded invocation of FP operations [15]. Nevertheless, the availability of idle CPU resources across the cluster creates opportunities to offload FP operations. Moreover, most cloud providers are now investing heavily in GPU equipped servers to handle compute-intensive workloads [16]. In the presence of GPUs in the cluster, the overhead of FP operations can be minimized, because of their high computing power, which is derived from their multi-threaded and many core parallel architecture [17], [18].

On the flip side, the current deduplication systems does not have a framework to utilize the computational power of idle CPUs or GPUs in neighboring servers across the cluster. Consequently, there is a need for an efficient communication framework and a dynamic resource scheduler that maximizes the use of available computational resources to enhance the deduplication performance. To build such a framework, there exist several challenges which need to be addressed. For example, (i) there is a need for a framework that provides a global view of the compute resources, that is, the distributed CPU and GPU resources in the cluster. However, the framework for providing a global view, when implemented in a centralized manner, not only raises scalability issues but also violates the design properties (shared-nothing) of the target architecture. (ii) The framework should be adaptive to dynamic load changing scenarios, to subdue under or over-utilization of compute resources. In this framework, reducing unnecessary data transfer and communication overhead is paramount.

To address such challenges, we propose CROCUS, a high-performance deduplication framework equipped with effective compute resource management, which provides a global view of the compute resources (CPU and GPU) available in the cluster. CROCUS offloads deduplication jobs to each of the pool based on load-awareness using a multi-stream communication model to enhance data transfer and to minimize communication overhead. The proposed framework is highly generic and can be adopted in both inline and offline deduplication with different storage architectures such as single and multi-tier storage setups.

This paper makes the following specific contributions.

- We build CROCUS, a cluster-wide deduplication framework for multi-tier hybrid storage architectures, capable of eliminating duplicates with negligible performance degradation. CROCUS provides an up-to-date view of available compute resources (CPU/GPU), without violating the design properties of shared-nothing storage systems.
- We design a simple yet dynamic Load-Aware Fingerprint Scheduler (LAFS), which is an efficient compute-resource orchestrator. LAFS parallelizes inline deduplication operations on available computing resources such as local or remote CPU and GPU across the cluster. It then distributes the compute-intensive deduplication operations to idle CPU and GPU resources in a load-aware fashion.
- We implement CROCUS in Ceph [3]. Our real testbed evaluation shows that CROCUS with LAFS scheduler improves the fingerprinting performance by 86 percent with 4 KB small chunks compared to a LAFS-free system. In addition, we achieve high-disk space savings using 4 KB small chunks with low performance degradation of 6 percent compared to Ceph without deduplication.

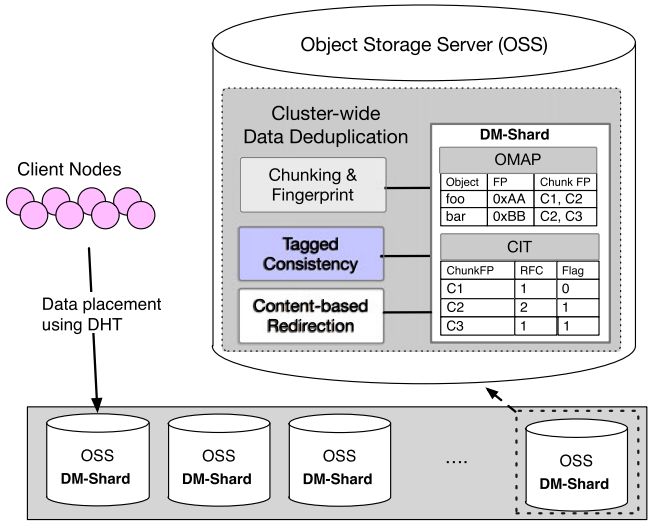


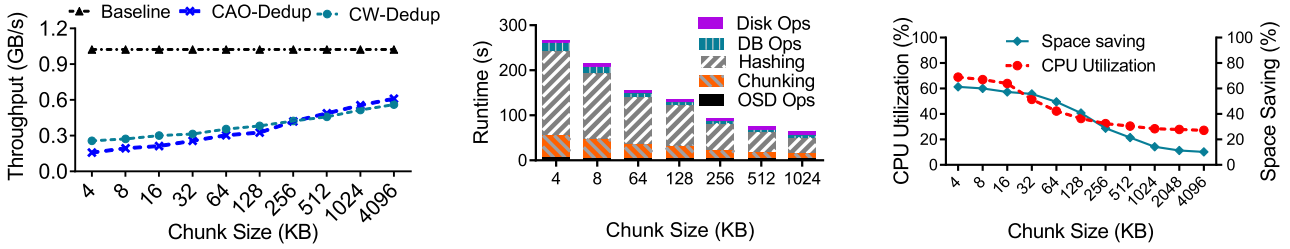
Fig. 1. Overview of inline cluster-wide data deduplication [11].

2 Background and Motivation

2.1 Cluster-Wide Deduplication

Cluster-wide deduplication has been widely explored to improve space efficiency in storage sytems. In general, deduplication is adopted either inline or offline. However, this design decision highly depends on the underlying storage cluster, as it directly impacts its performance. For instance, inline deduplication offers instant space savings but at the cost of performance degradation as it includes deduplication operations such as chunking, hashing and duplicate lookup I/Os in the critical I/O path. On the contrary, offline deduplication also referred to as lazy deduplication requires a temporary data staging buffer space to store data. Note that, the offline approach does not include deduplication operation in the I/O critical path but incurs a double write problem, that is, i) all incoming data is written to the temporary staging buffer space and ii) then after applying deduplication, unique data is written to permanent storage again. This double write problem is harmful when the storage used is SSDs. Moreover, offline deduplication requires certain controls, e.g., a size threshold on staging buffer space to trigger deduplication, which often conflicts with incoming write I/Os. As our target system is a shared-nothing tiered hybrid architecture, where SSD servers overlay slow HDD based storage for higher performance, deploying inline deduplication provides the best choice due to limited SSD storage endurance and higher cost ($/GB) issues.

Although the cluster wide deduplication has been explored in several studies [11], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], these contemporary deduplication techniques cannot be directly adopted because of two major reasons; (i) They do not comply with the shared-nothing



(a) Performance comparison (b) Time breakdown analysis (c) CPU utilization and space savings

Fig. 2. Performance analysis of cluster-wide deduplication when employed on hybrid Ceph cluster. We used Testbed-1 with SSDs, HDDs, and inactive GPUs.

design property of the target architecture. (ii) They were not designed to work effectively on scalable tiered hybrid data storage model. There exist two most recent studies which proposed cluster-wide data deduplication without violating the design properties of shared-nothing storage systems, [11], [12]. The work in [11] manages the deduplication metadata using the partitioned database approach (DM-Shard), where each horizontally partitioned database instance (also called DB-Shard) is tightly integrated to each object storage node in the cluster as shown in Fig. 1. Each DM-Shard stores two persistent data structures, i.e., Object Map (OMAP) containing layout information of objects and Chunk Information Table (CIT) containing sensitive deduplication metadata such as chunk fingerprint, reference count and consistency flag. On the other hand, [12] proposed to handle the deduplication metadata in the form of a content addressable object (CAO), which is stored just like any other object in the underlying storage. In order to access the metadata information, the use of the extended attribute (xattr) is adopted. Both approaches assured conformity to the shared-nothing design constraint.

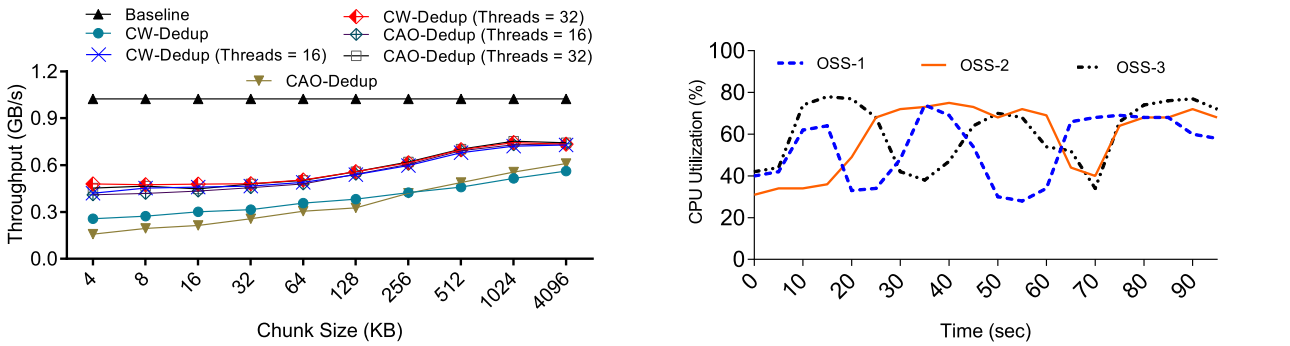
2.2 Motivation

To motivate our research, we first investigated the adoption of the two recent inline cluster-wide deduplication techniques, compatible with the shared nothing design storage systems (CW-Dedup implementation [11] and CAO-Dedup implementation [12]), on top of a hybrid tiered Ceph cluster. We configured the Ceph cluster by using Testbed-I with no active GPUs as shown in Table 1 (Section 5). We used the fixed-size chunking algorithm and SHA-1 hash function to compute the hash fingerprints for each data chunk. Note that other content defined chunking methods such as FASTCDC [14] can be applied to CROCUS. However, this work focuses on overcoming the shortcomings of existing deduplication frameworks. As workloads for evaluation, we generated a total of 2 TB writes per client via the SPEC SFS benchmark [29]. For this particular experiment, we used 5 client nodes each hosting 16 threads issuing write IOs. Note that we disabled GPUs on all the nodes of Testbed-I to clearly investigate the influence of deduplication on performance. We issued random write requests to the Ceph cluster and on each successive experiment, object size was fixed to 4 MB but doubled the chunk size by 2. The results in Fig. 2 show the outcome of our investigation on the existing inline cluster-wide deduplication techniques.

Fig. 2a, shows that adopting the current deduplication implementations [11], [12], introduces a huge performance degradation as compared to the baseline without deduplication. This performance penalty is worse with small-sized chunks. To investigate the cause of the performance overhead, we carried out a time breakdown analysis on CW-Dedup [11], for a single dedup IO. Fig. 2b shows the results. CAO-Dedup [12] shows high performance penalty as compared to CW-Dedup [11] due to the additional metadata checks of CAO objects on storage. So, we focused our investigation of the performance penalty on CW-Dedup. Fig. 2b shows the aggregated runtime for each deduplication operation for the whole dataset. We observed that the overhead of computing fingerprints is considerably large. It contributes to more than 65 percent of the aggregated runtime. Because the current implementation uses a single thread to invoke hash operations, the overhead of fingerprinting is quite high and a more robust approach is required to reduce this performance penalty. Fig. 2c shows the CPU overhead and storage space gain in the object storage server depending on the chunk size. The smaller chunk size benefits with high space savings. However, small chunk sizes show high CPU utilization which averages 65 percent across the storage servers. On the contrary, the large chunk sizes result in lower CPU occupancy but at the cost of low disk space savings. Even though the current deduplication implementation is single threaded, but fingerprinting operation shows quite a high CPU utilization especially with the smaller chunks which achieves more deduplication benefits.

2.3 Deduplication Improvement Opportunities

Since the current cluster-wide deduplication implementations uses a single thread fingerprinting approach, opportunities to improve the performance can arise with multi-threaded fingerprinting invocations. Fig. 3a shows the results of employing multi-thread invocations of the hash function to compute fingerprints on CW-Dedup and CAO-Dedup. We observe that, for both implementations, using 16 threads will increase the throughput, but the performance is still degraded as compared to baseline with no deduplication. Nevertheless, increasing to 32 threads does not improve any performance. This is because fingerprint computation is a CPU bound operation. In scale-out storage systems, there are frequent object storage daemon operations and continuous internal communication which enforces replication, load re-balancing and node heartbeat messages. This generates high amounts of CPU load and suppresses the performance improvement from the multi-threaded dedup approach. In our case, the performance saturates at 16 threads. However, in distributed clusters, there exist pockets of CPU idle times or low CPU



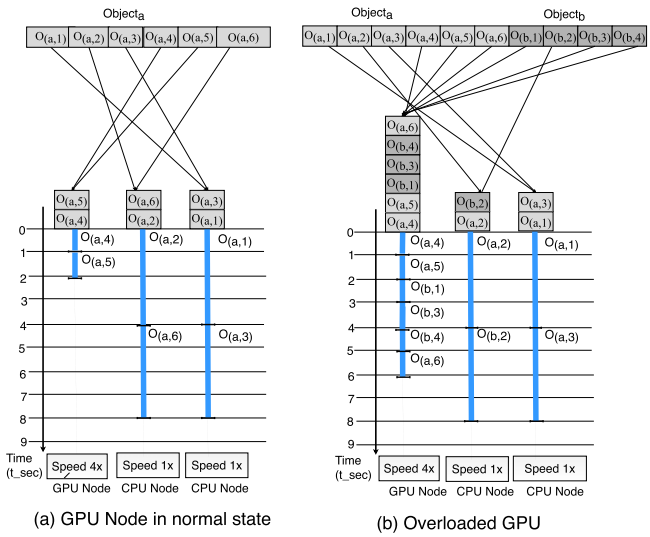
(a) Multi-threaded deduplication analysis (b) CPU utilization snapshot on SSD tier Object Storage Servers (OSSs)

Fig. 3. Improving the deduplication performance via multi-threaded deduplication.

utilization on object storage servers (OSSs). Fig. 3b shows a snapshot of the CPU utilization of the SSD tier nodes on our Ceph cluster, using 4 KB chunking, over a period of 90 seconds during a write request to the cluster. These pockets of low CPU utilization shown in the snapshot provide opportunities to use computing resources on neighboring storage servers when a server has no available computational resources.

Futher, if cluster nodes are GPU equipped, then we can leverage the massive parallelism provided by GPUs to accelerate the overhead of fingerprint computations. There are two scenarios to adopt when setting up GPU nodes on the faster SSD tier. One case is a homogeneous GPU-equipped SSD tier and the other case is when not every server on the SSD tier is GPU-equipped. However, the second case creates computational heterogeneity on the SSD tier. This heterogeneity leads to some bottleneck, since the CPU based fingerprinting of non-GPU servers slows down the I/O performance. We explain this problem using Fig. 4, where object_a is striped into multiple smaller objects $O_{(a,1)}$ to $O_{(a,6)}$ and written to different SSD tier nodes for deduplication. In Fig. 4a, we use an example where deduplication on GPU node is 4x faster as compared to CPU node. Let t be the time unit to compute one object chunk. The time to compute deduplication of object $O_{(a,4)}$ and object $O_{(a,5)}$ placed on the GPU node will be $2t$. However, the time to compute the same number of objects placed on the other 2 CPU nodes will be 8t. This entails that the overall deduplication time for Object_a is determined by the tail dedup operations on slower CPU nodes. Assuming negligible network overhead, offloading objects $O_{(a,1)}$, $O_{(a,2)}$, $O_{(a,3)}$ and $O_{(a,6)}$ onto GPU node will reduce the overall deduplication time from $8t$ to $6t$. This shows the need to offload slower deduplication processes from CPU to GPU node.

However, there are situations where the GPU node is overloaded (depicted by Fig. 4). In this case, blindly offloading deduplication computations to the GPU may result in worse performance. For example, In Fig. 4, where two files, Object_a and Object_b, from two different client nodes, are written to the SSD tier nodes. We see that when normal scheduling is done for the two objects, a and b, the processing time will complete in time 8t. However, if we consider offloading all the objects ($O_{a,1}$, $O_{a,2}$, $O_{a,3}$, and $O_{b,2}$) from CPU nodes to the GPU node, just because of its higher computational power, each object ($O_{a,1}$, $O_{a,2}$, $O_{a,3}$, and $O_{b,2}$) will add a processing time of 1t to the GPU node, hence increasing the total processing time from 8t to 10t. Therefore, there is a need for CPU nodes to dynamically schedule the amount of data to offload to GPU nodes based on the workload intensity of GPU nodes.



(a) GPU Node in normal state (b) Overloaded GPU

Fig. 4. Illustration of performance degradation due to redirection without consideration of GPU load.

To this end, we are motivated to propose CROCUS, a holistic approach which orchestrates the distribution of deduplication processes to all participating compute resources (local/remote CPU and GPU nodes).

3 CROCUS ARCHITECTURAL OVERVIEW

The primary design goal of CROCUS is to build a high performance inline cluster-wide deduplication framework, capable of minimizing the performance degradation introduced by deduplication operations. These operations include chunking and fingerprinting at the SSD tier of the tiered storage system. Fig. 5 presents the architectural overview of CROCUS. The SSD and HDD tiers depict the hybrid nature of scale-out storage system. Importantly, only SSD tier nodes are equipped with CROCUS as shown in Fig. 5 whereas, HDD tier is simply responsible for storing the unique data.

The client nodes perform object name hashing via DHT-based algorithm such as CRUSH in Ceph [3] to place or retrieve objects in the cluster. Due to the tiered architecture, the object placement location is computed to one of the SSD tier nodes. Once the I/O lands on an SSD tier node, CROCUS

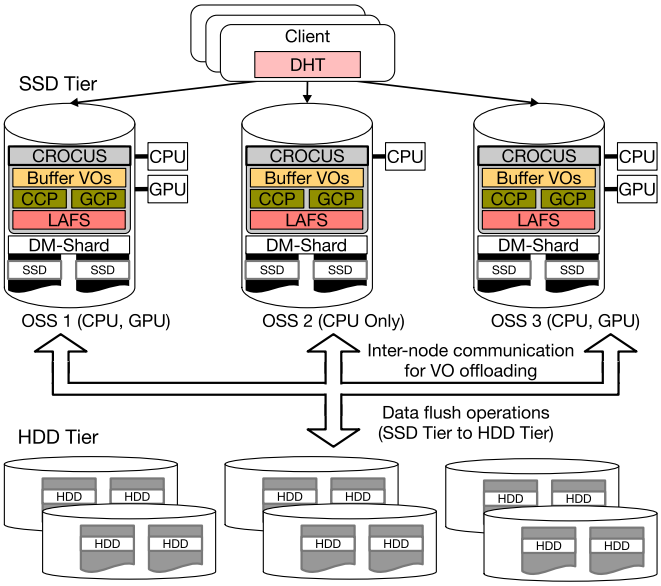


Fig. 5. Overview of Crocus architecture.

divides the objects into smaller fixed size chunks and is required to compute the fingerprint using a cryptographic hash function such as SHA1 or SHA512. In this study, we chose fixed size chunking to illustrate the benefits of Crocus. However, Crocus is also applicable for variable size chunking. The direct fingerprint computation using local CPU is not optimal and might degrade the deduplication performance. Crocus is equipped with an intelligent, load-aware fingerprint scheduler, which offloads the fingerprint operations to compute resources such as local and remote CPU or GPU based on the availability of each resource. In order to use remote resources while avoiding network overhead, Crocus batches several objects into a single large object unit, which we call virtual object (VO) and offloads it onto available CPU or GPU resource for effective fingerprint computation.

Crocus manages resource availability through maintaining two data structures globally across each SSD tier node, i.e., CPU Computation Pool (CCP) consisting of all SSD tier nodes with CPU (as it is obvious that all nodes are equipped with CPU) and GPU Computation Pool (GCP) consisting of SSD tier nodes equipped with GPU accelerator units. These data structures not only assist in optimal fingerprint computation but also empower Crocus to handle resource heterogeneity, if any, among SSD tier nodes. In cases of this resource heterogeneity, where not every SSD tier node is equipped with GPU accelerators, client nodes are unaware of such GPU availability on SSD tier nodes. Worse off, clients can issue write I/Os directly to SSD tier nodes without GPU hardware. The details of VO, LAFS, and data structures are provided in Section 4.

Once fingerprints are computed against the VOs then, deduplication related metadata is updated in the local Deduplication Metadata Shard (DM-Shard). After deduplication metadata operations, unique data chunks are redirected for real-time storage and I/O transaction finishes. Note that client immediately receives an ack message when dedup metadata is successfully added in DM-Shard. This reduces the additional latency in I/O path. In order to ensure data and metadata consistency, we rely on the asynchronous tagged-consistency provided by [11], which provides for eventual consistency for the chunks with updated metadata but not written on disk.

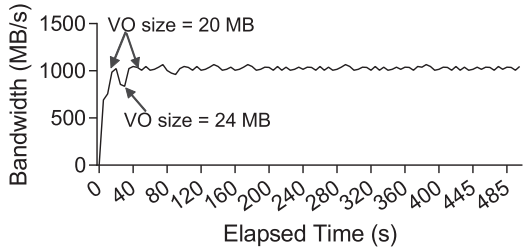


Fig. 6. Automated VO re-sizing.

4 Design and Implementation

4.1 Compute Resource Pooling

To devise a resource and load-aware fingerprint scheduler in Crocus, we formulate our model as a compute-resource mapper, which takes into account several thresholds and constraints. The threshold includes buffering of objects up to a certain limit at each SSD tier node, and constraints such as inline deduplication and minimizing additional latency in fingerprint computation are considered. We achieve this by characterizing the available compute resources into a vector pair consisting of two pools, i.e., CCP and GCP. All the SSD tier nodes along with their CPU utilization ratios are managed by the CCP, whereas the GCP is responsible for maintaining the GPU utilization ratios of GPU equipped SSD tier nodes.

In GPU heterogeneous SSD tiers, it is possible that the number of nodes in CCP can be a magnitude higher than the number of nodes in GCP, which motivates us to consider neighbor CPU/GPU resources in our model formulation. We observed from Fig. 3b, that, there exist opportunities to exploit idle CPU resources residing on the SSD tier nodes to amortize the fingerprint computation cost. Both CCP and GCP maintain a list of SSD tier nodes sorted based on each resource availability with respect to the number of admissible VOs for service.

4.2 Virtual Object and Partition Creation

Our model uses the Virtual Object as a basic unit to offload or schedule a fingerprint operation on remote SSD tier nodes. Each VO consists of multiple objects batched together. We set the VO size as a tunable parameter for any cluster setup. However, a challenge arises on how to determine the optimal VO size for any given cluster configuration. In order to overcome this problem we designed an automated VO resizing strategy that optimizes the best VO size for a given cluster hardware configuration.

To choose the optimum VO size, Crocus first initiates the VO size equal to the original object size, i.e., 4 MB, and then increases the VO size by a single object after every 5 seconds. Fig. 6 shows how the automated VO resizing works. After 20 seconds, when the VO size would have adjusted to 20 MB, Crocus reaches an optimal performance. Increasing the VO size after that degrades the performance. This degradation is reflected in Fig. 6 after 25 seconds when the VO size is adjusted to 24 MB, in which the performance drops by approximately 12 percent.

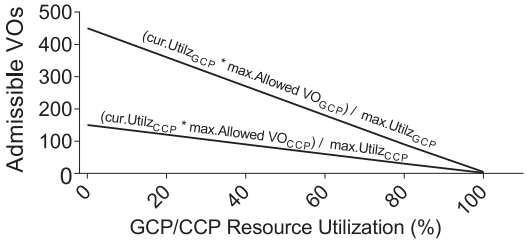


Fig. 7. Admissible number of VOs in GCP and CCP with changing utilization ratio of each compute pool.

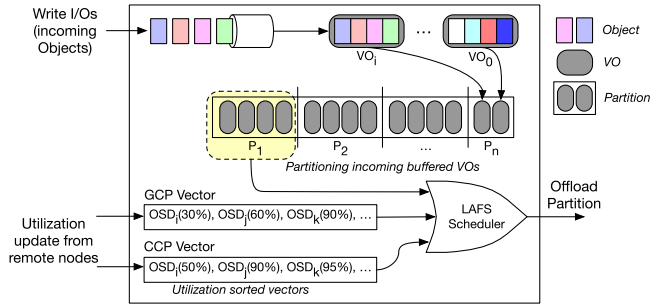


Fig. 8. Partition scheduling in CROCUS LAFS scheduler.

When there is a performance drop of more than 5 percent, CROCUS reverts the VO size to the previously allocated VO size with best performance. We set this 5 percent value to minimize the number of VO size adjustments. After the adjustment point CROCUS maintains the VO size as optimum. The CCP or GCP then permits scheduling of VOs as per available compute resource bandwidth.

For example, every GCP node computes the admissible VOs for scheduling using the following equation;

$$(cur.utilz_{GCP} * max.allowedVO_{GCP})/max.utilz_{GCP}. \quad (1)$$

Similarly, all CCP nodes compute the number of VOs to be scheduled using the following equation;

$$(cur.utilz_{CCP} * max.allowedVO_{CCP}) \ /max.utilz_{CCP}. \quad (2)$$

Cur. utilz in equations denotes the currently available GPU and CPU nodes utilization, which is computed simply as 100 percent - CPU/GPU utilization. *Max. allowed VO* is the maximum number of VOs admissible to CPU or GPU resources (This is the case when the GPU or CPU resources are free) and *Max.utilz* is maximum utilization ratio of GPU or CPU resources. The most important parameter is the *max.allowed VO* which denotes the maximum number of VOs admissible to the GCP and CCP nodes when their utilization is close to zero. This number is specified in Section 5.1 under system parameters. In the above modeling, the max. utilz and max.allowed VO are constants and the number of admissible VOs are therefore determined by the cur.utilz value which represents the varying resource utilization ratio at any given time.

We observe that the number of admissible VOs at any given instant is directly proportional to the available GCP/CCP node compute resource utilization ratios. Fig. 7 reflects the results of our VO admission model, which shows a linear decrease in the VO intake of both the GCP and CCP nodes with increasing resource utilization of the nodes. The results of this model motivates us to prioritize the GCP pool nodes due to their higher VO admission rates. Finally, after computing the permissible number of VOs, each compute pool node, i.e., CCP or GCP nodes periodically broadcast its remaining number of VO intake or current utilization ratio. However, with the co-location of GPUs and CPUs in the storage nodes, CROCUS prioritizes scheduling of local VO buffers to local GPU over remote VO buffers as they execute faster. CROCUS differentiates the local and remote buffers due to local DMA tagging and remote RDMA tagging.

We allocate a certain memory buffer area on each SSD tier node, which we called a Partition. All incoming data objects from clients are staged for fingerprinting scheduling in this partitioned area as shown in Fig. 8. Each partition consists of VO_1 up to VO_n. We avoid scheduling based on individual objects as it increases the node to node data transfer and introduces communication overheads. Once a VO is scheduled to a remote node, it will be treated as another incoming object on the remote node by default. Hence, the scheduled VO can be rescheduled again to another node. In this way, the VO can continue to be rescheduled to other nodes. This phenomenon is called a circular scheduling problem. We avoid this circular scheduling problem by separating scheduling/partition buffers from the deduplication process buffers. Consequently, the scheduled VO only land in the deduplication process buffers.

To this end, we have identified a few system tunable parameters on which the performance of LAFS can vary. These parameters include, (i) maximum admisable VOs for CPU/GPU pools of each SSD tier node and (ii) resource utilization vector update interval. These parameter values depend on workload and hardware characteristics. These values are set as configuration parameters and can be tuned to the required default settings. Therefore, finding the optimal values for these tunable parameters is not the scope of this study.

4.3 Dynamic Load-Aware Fingerprint Scheduler

The design motivation behind dynamic load-aware fingerprint scheduler includes the efficient distribution of deduplication operations, such as chunking and fingerprinting, across all available compute resources on the SSD tier nodes. This eradicates the idle times of compute resources in all nodes on the SSD tier, hence achieving maximum resource utilization. LAFS is the core scheduling engine embedded on each SSD tier node, responsible for cluster-wide data deduplication. LAFS makes scheduling decisions on incoming VOs as shown in Fig. 8. Then, with the utilization ratios provided by the GCP and CCP pools, the LAFS scheduler assign the VOs in the current scheduling partition, by referencing the current utilization ratios of compute resources in GCP and CCP. This mapping prioritizes GCP over CCP due to the higher throughput and performance of GPUs. If no GPU-equiped nodes are present in the cluster, LAFS directly schedules the VOs onto the CCP pool nodes. Recalling, both GCP and CCP maintain a vector of weighted nodes in a sorted manner of respective compute pools.

When two remote SSD tier nodes have an equal current utilization ratio, LAFS scheduler randomly selects the node to schedule among the two. In case the list of GCP pool is fully exhausted, LAFS shifts the distribution of VOs to the CCP pool nodes. Note that if remote nodes fail whilst

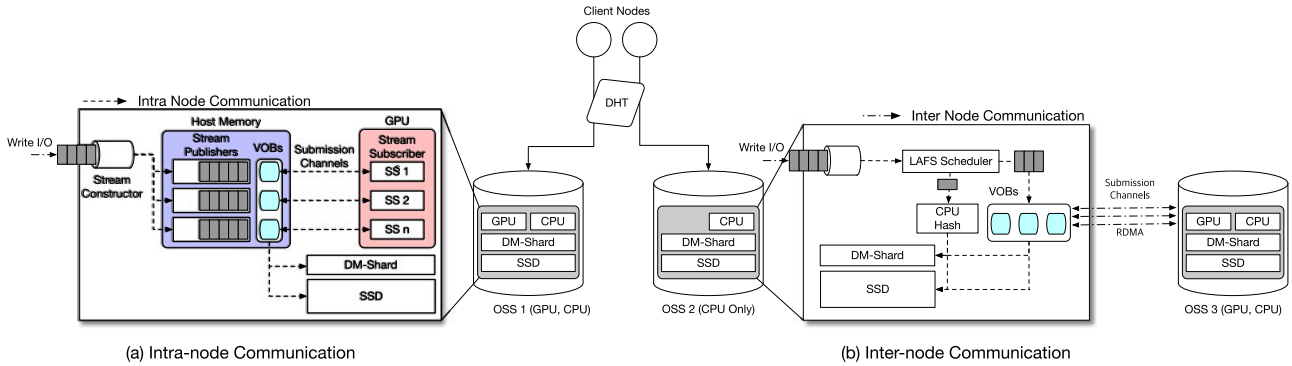


Fig. 9. a) Crocus intra-node communication optimization in GPU-equipped nodes and (b) inter-node scheduling mechanism in CPU nodes.

executing fingerprint operations, metadata would not have been updated. This makes our system free from such failures which might cause any inconsistencies, since we push metadata to persistent storage only when transaction completes. Another critical but rare case which is highly dependent and affected by vector update interval is that, two or more nodes can simultaneously offload VOs to a remote node which cannot facilitate both VOs. This results in the over-scheduling of VOs to that remote node. To cushion such bursty VO offloading cases, we set an abstracted over-provisioned buffer area in all nodes, which provides some elasticity in these over-scheduling scenarios.

4.4 GPU and Communication Optimization

To maximize inline deduplication throughput on GPU equipped nodes, i.e., GCP pool, efficient data transfer techniques and effective memory management approaches must be employed on both CPU host and GPU acceleration unit. In particular, when using local/remote GPU acceleration units, we aim to eradicate the serialization dependency between data transfers and the GPU computations that occur in the default GPU computational model. We implement multi-stream communication channels between CPU memory and local/remote GPU memory as depicted by Fig. 9. The communication protocol is composed of a Stream Constructor (SC) which constructs stream channels from incoming data. All GPU threads executing in a specific stream are called Stream Subscribers (SS) and the corresponding CPU mapped memory per stream is called the Stream Publisher (SP.) This multi-stream design enables interleaving of deduplication tasks running on GPUs as much as possible, thus maximizing the high parallelism offered by GPUs.

We implemented a sliding window chunking and fingerprinting kernel, which computes fingerprints from the beginning of GPU memory address space of each stream. Depending on chunk size specification, SS threads slide by offset positions equal to the chunk size whilst calculating the fingerprint of each chunk. Once FPs for all the chunks in the stream are computed, the FP values are transferred back to host memory and stream is then destroyed clearing up buffer area for new streams to be created. Fingerprint lookup is then issued to respective DM-Shard to validate if the data chunks existence in storage or not.

Intra-Node Multi-Stream Offloading of VOs to GPU via DMA Operations. This occurs when data is placed on a GPU-equipped node and LAFS schedules to use its node local GPU, as depicted by Fig. 9a. When objects are ingested into GPU equipped node, the stream constructor initiates equal sized streams. Each stream is created one after the other with a fixed-size equal to the configured size of the partition. The stream constructor forwards them to a temporary fixed buffer area of host memory called the stream publisher. Upon data request by stream subscriber threads, VOs are then transferred to GPU memory. The SS threads always fetches data from the corresponding SP and this mapping is leveraged by the GPU driver.

Inter-Node Multi-Stream Re-Routing of VOs to Remote GPU Node. This case is triggered when a CCP node schedules chunking and fingerprinting operations on a remote GCP node, as depicted by Fig. 9b. There are two possible ways of inter-node data transfers. The default CUDA inter-node protocol follows a pipelined approach that first copies data between host memories of the two nodes (H2H) over the network, then once data is on the GPU equipped node, it is offloaded to GPU device memory via additional cudaMemcpy operations for accelerated deduplication processes. The second option, which we used for our implementation is for the CPU node to directly communicate with the remote node GPU via the network using RDMA transfers. Direct $\mathrm{H2D}_{\mathrm{remote}}$ and $\mathrm{D2H}_{\mathrm{remote}}$ reduce the additional latencies caused by proxy processes.

Inter-Node Multi-Stream Re-Routing of VOs to Remote CPU Node. This is when a node is not GPU-equipped, and all remote GPU nodes are overloaded and the LAFS scheduler instructs to use a remote CPU node. In this case, using the MPI-based communication API allows two CPU nodes to exchange VOs. In addition, if the RDMA network is available, it performs RDMA-based MPI communication.

4.5 Crocus Overhead

Recall that Crocus achieves maximum deduplication throughput by utilizing idle compute resources of either Local GPU or neighboring GPU/CPU nodes. However, offloading objects to other nodes in the cluster may incur additional network overheads thus increasing the I/O latency. However, executing deduplication with CPU nodes cause more performance bottleneck due to the limitations of CPUs described in Section 2.2, which restrains maximum deduplication throughput. In our proposed approach, we argue that although there is a small increase in latency per I/O, there is a significant throughput

TABLE 1
Experiment Testbed

Testbed	SSD Tier	HDD Tier
Testbed-I	3 Nodes, 2 SSDs Per Node, 1 GPU Per Node	3 Nodes, 2x1 TB ATA HDD Per Node
Testbed-II	7 Nodes (2 GPU Nodes, 5 CPU Nodes), 2 SSDs Per Node	5 Nodes, 2x1 TB ATA HDD Per Node

performance boost achieved from offloading deduplication tasks to faster GPU nodes and idle CPU node resources. We also amortize the network overheads through a batching operation that groups multiple objects into a single larger VO size that will be used as a unit for data transfer to neighboring nodes. We discuss the network performance trade-off in Section 5.4.

5 Evaluation

5.1 Experimental Setup

Our experimental setup consists of a Ceph Cluster with two tiers to depict a hybrid tiered storage model. The testbeds used for evaluation are shown in Table 1. In Testbed-I, all SSD tier nodes have equal resource configurations, whereas in Testbed-II only 2 nodes are equipped with GPU hardware while others are not GPU equipped to depict compute resource heterogeneity on the SSD tier. Each node runs Linux CentOS v7.3 and equipped with 10 cores Intel Xeon (R) CPU running at 2.40 GHz with a 32 GB DRAM. The nodes with GPU on SSD tier are provisioned with Nvidia Tesla K80, 24 GB GPU. On the storage side, each SSD tier node has two 256 GB Intel 850 PRO SSDs and HDD tier nodes have two 1 TB Hard disk drives, all connected via a 10 Gbps network. To flush the unique chunks from the SSD to the HDD tier, we set the cache-min-evict-age to 1800 ms, which evicts the chunks every 30 seconds and provided for the optimum flushing performance.

Workloads. We used both synthetic and realistic benchmarks to evaluate Crocus. For synthetic workloads, we used FIO benchmark [30] and for realistic workloads, we used SPEC SFS 2014 database benchmark [29]. For SPEC SFS workload, we mounted a block device on each client node and generated a total of 2 TB files per client node using a total of 5 clients. We then issued random write requests to the Ceph cluster. To fairly evaluate the effectiveness of LAFS in high contention scenarios, we run the SRAD application from the Rodinia Benchmark suite [31] to vary the load in GPU nodes. The LAFS scheduler in such cases is forced to redistribute the fingerprint requests between CPU and GPU pool nodes in an adaptive manner based on the available compute resources in the nodes. All the results in our evaluation show an end to end aggregate time or throughput for all deduplication operations, for the whole dataset and for all approaches. Our results show an average of three experimental runs. We compare the following systems:

- No-Dedup: Original Ceph with no deduplication. We use baseline and No-Dedup interchangeably in the rest of section.
- Dedup-CPU: Ceph with inline cluster-wide deduplication on SSD tier using CPU with single-thread implementation.
- Dedup-CPU(m): Ceph with inline cluster-wide deduplication on SSD tier using CPU invoking multi-threaded (16 Threads) hash computations.
- Dedup-GPU: Ceph with inline cluster-wide deduplication [11], on SSD tier using GPU with no optimizations.
- Crocus: Ceph with inline cluster-wide deduplication on SSD tier using GPU with intra/inter-node optimizations.

System Parameters. We set the object size to 4 MB, which is the default in Ceph and the VO size is determined by the automated VO resizing approach (in our case 20 MB). Note that the performance varies with chunk and VO size but not object size. Irrespective of object size, the chunking is conducted and VOs are the scheduling units. Smaller VO size will require more aggressive scheduling and communications. The maximum allowed VOs on GCP are 440 and 150 VOs for CCP. We extract these numbers after performing a set of experiments whilst monitoring congestion levels in GCP and CCP. We set the periodic broadcast time of the GCP and CCP pool vectors to every 5 seconds. We obtain this broadcast time after running the experiment shown in Section 5.3.1. We use same system parameter values for all experiments unless specified.

5.2 Crocus in Homogeneous Cluster Settings

To show the effectiveness of offloading deduplication operations such as chunking and fingerprinting, we first analyze the deduplication performance with CPU only nodes and later with all GPU-equipped nodes. We perform the experiments with SPEC SFS workload and instrument a trace for each deduplication sub-operation that measures the time of each sub-operation separately. Fig. 10 shows that the Dedup-CPU (DC) constitutes an average of 75 percent runtime only for deduplication operations, i.e., chunking, fingerprinting and fingerprint lookup (DB Ops). This huge overhead is mainly attributed to fingerprinting and it gets worse with small chunk size.

On the other hand, Dedup-GPU (DG) reduces the deduplication overhead by 86 percent on the total runtime for smaller chunk sizes such as 4 KB. It is due to offloading the fingerprint computations onto the GPU. Transferring data across the GPU PCI-e bus in the unit of a chunk causes a significant overhead when the chunk size becomes small. However, CROCUS design avoids this by transferring batches of data objects (VOs) and chunking initiates once the VOs land on the GPU global memory. Thus, the chunk size does not have an effect on the transfer time over the GPU PCI-e bus. However, the important thing to note here is that Dedup-GPU slightly increases the fingerprint runtime with increasing chunk size. It is because when the chunk size is small, it precisely fits in the fast GPU shared memory and hence the sliding window of the fingerprint kernel can quickly shift through the offset positions of smaller chunk boundaries. Whereas, for the large chunk size, the fingerprint kernel threads shift many times, fetching data between the small but fast GPU shared memory and the larger GPU global memory before shifting to another chunk boundary. This result shows that the use of GPU can greatly reduce the

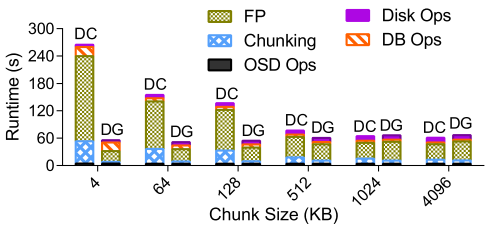


Fig. 10. Time breakdown analysis. Dedup-GPU (DG) versus Dedup-CPU (DC).

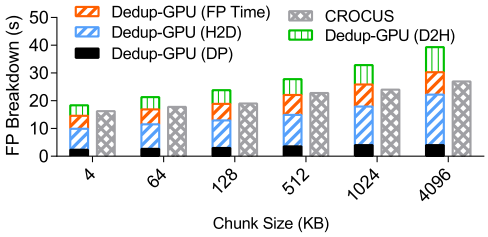


Fig. 11. Effects of multistream interleaving using per stream VO offloading.

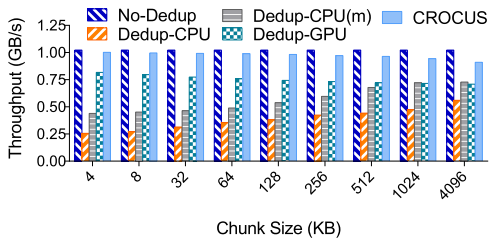


Fig. 12. Performance evaluation with Realistic SPEC SFS workload.

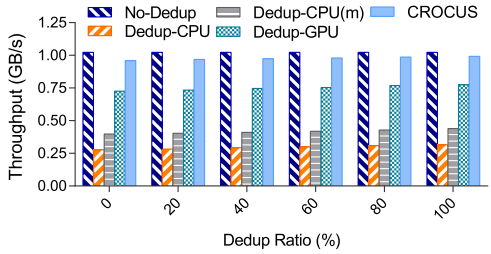


Fig. 13. Performance evaluation with synthetic FIO workload.

overheads of deduplication. However, in a heterogeneous setting it might not be the case as some SSD tier nodes will not be equipped with GPU. We evaluate how Crocus deals with such heterogeneity in Section 5.3.

To achieve the best deduplication performance with GPU in an all GPU SSD tier, we proposed some inter-node multi-stream VO offloading. In this section, we show the significance of our proposed intra-node optimizations with respect to varying chunk size. We compare the proposed intra-node optimizations (Crocus) against Dedup-GPU provisioned with no GPU optimizations. Fig. 11 compares the results of the optimizations versus non optimized GPU-Dedup. We trace the overall time for data preparation (DP), data transfers (Host-to-Device and Device-to-Host), and fingerprint kernel execution time in Dedup-GPU. However, the overall fingerprint computation time for Crocus consists of an overlap of data transfers and FP executions between the multiple stream submission channels, so we only collect the overall runtime. The results show that the multi-stream processing of Crocus improves the fingerprint runtime by 33 percent compared to Dedup-GPU. This improvement is attributed to (i) overlapping data transfers and fingerprint kernel executions and (ii) interleaving multiple fingerprint computations from different stream submission channels.

5.2.1 Crocus for Various Workload Characteristics

To evaluate Crocus with different workloads in a homogeneous SSD tier setup, we used synthetic and realistic workloads. We tried to mimic different deduplication attributes such as varying chunk size and duplication ratio. For realistic workloads, we employ SPEC SFS [29] and use FIO [30] to generate synthetic workload with controlled deduplication ratio. We compare Crocus with different variants of deduplication implementations. Dedup-CPU(m) in Figs. 12 and 13 employs multiple threads instead of a single thread. Fig. 12 shows the performance trend with varying chunk size using SPEC SFS workload. The results shows that both Dedup-CPU versions degrade performance specifically with small chunks. On the contrary, both Dedup-GPU implementations outperform the CPU implementations because of offloading compute-intensive operations on the powerful GPUs. With small 4KB chunk size, Dedup-GPU reduces the overall performance degradation from a high of 76 percent (in Dedup-CPU) to a significant low of 32 percent and Crocus further minimizes to the best low of 6 percent. Crocus performs better than Dedup-GPU due to the less synchronous communication between host and device, through the parallel transfers and fingerprint computations on different stream channels. However, we observe a minor performance degradation while traversing from smaller to bigger chunk sizes in GPU implementations. This is because, an increase in chunk size increases the number of shift cycles between the global and shared memory of the GPU.

Next, we show the effect of deduplication ratio using FIO generated workload in Fig. 13. For this experiment, we fix the chunk size to 32 KB (less than shared memory size of 48 KB on the Tesla K80 GPU) and vary the deduplication ratio. We observe that with increasing deduplication ratio from 0 to 100 percent, performance also slightly improves in all implementations, i.e., Dedup-CPU, Dedup-CPU(m), Dedup-GPU and Crocus. It is because the number of duplicate fingerprints increases with increasing dedup ratio, consequently, the number of I/O requests to the underlying storage reduces. However, the performance of Crocus still outperforms the Dedup-GPU, Dedup-CPU and Dedup-CPU (m) implementations. This improvement is attributed to proposed multi-stream communications in Crocus.

5.3 LAFS in Heterogeneous Cluster Settings

In this section, we perform a set of experiments to show the effectiveness of the proposed LAFS scheduler in Crocus. We use Testbed-II to incorporate resource heterogeneity on SSD tier nodes as shown in Table 1. To the best of our knowledge, no prior work has been done to explore deduplication scheduling orchestrations across the entire cluster. In this regard, we compare our dynamic LAFS of Crocus with a static scheduling approach. The static scheduling

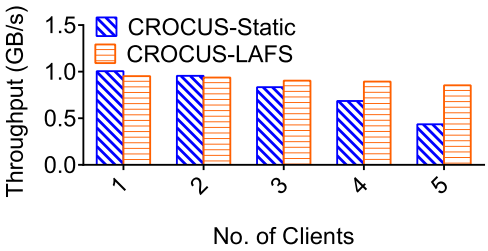


Fig. 14. Static versus dynamic scheduling.

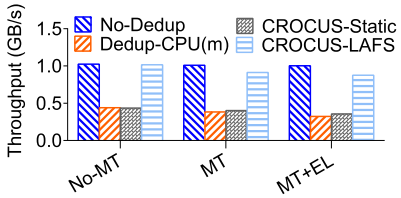


Fig. 15. Load-aware LAFS.

approach schedules all fingerprint operations to GPU nodes without the knowledge of their load or contention. It performs well under normal circumstances, but, degrades performance when the load on GPU nodes increases. However, the proposed dynamic LAFS scheduler adapts well to dynamically changing workloads. We perform the experiment with a single client and then, add more clients on each experimental run till we reach five clients whilst observing the impact of increasing load on LAFS scheduling. Fig. 14 shows a nearly equal performance for both LAFS and static scheduling with a few number of clients due to less I/O contention on GPU nodes. However, the LAFS scheduler significantly outperforms the static scheduler with increased number of clients. The reason of the low performance of static scheduling is the high number of scheduling operations leading to contention on GPUs. Whereas, the LAFS scheduler is aware of workload changes on GPU and can dynamically adjust the workload distribution between CPU and GPU nodes, thus outperforming the static scheduling approach.

For the next experiment, we mimic the realistic scenarios and evaluate the performance of LAFS scheduler by increasing the number of threads on each client node (total of five physical client nodes), to the storage cluster. Also as cloud environments hosts several VMs running multiple applications on a single node, some VM applications make heavy use of GPUs than others. To mimic this scenario, we further stress GPU nodes by generating additional load to GPUs via Rodinia SRAD Benchmark application [31], whilst monitoring the LAFS performance. Fig. 15 shows this experimental results. The No-MT stands for 5 client nodes with no multi-threading, MT stands for 5 client nodes each running 32 threads to continuously issue I/O requests, and MT+EL stands for 5 client nodes, each running 32 threads with external load (EL) generated via SRAD application. The results show that when there is no contention (No-MT), Crocus-LAFS performance is nearly at equal to No-Dedup. Whereas, Crocus performance drops sharply when integrated with static scheduling approach. This drop is worse off when the external load on GPU nodes increase as depicted in MT+EL case. However, the point here is that

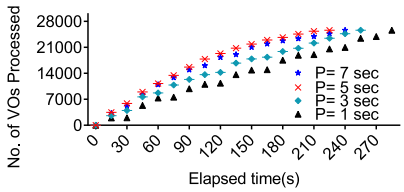


Fig. 16. Sensitivity test of vector updates.

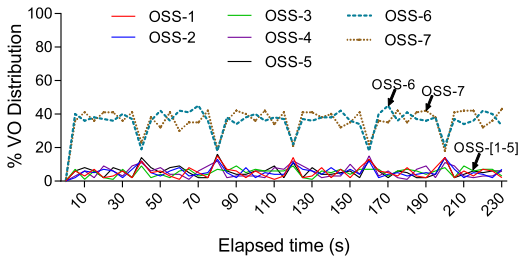
Crocus-LAFS shows a nearly consistent performance for all cases, even in cases of high contention on GPU nodes by distributing the workload on CPU and GPU nodes proportionally to available resource bandwidths.

5.3.1 Sensitivity Test and VO Distribution Pattern

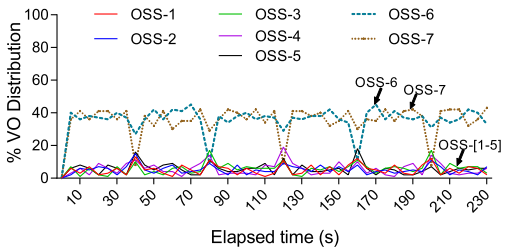
The LAFS scheduler highly depends on utilization ratios of compute resources to schedule or offload fingerprint operations on SSD tier nodes. Thus, the utilization vector update interval is very critical. We perform a sensitivity test to analyze how this update interval impacts the number of VOs processed per unit time. We performed this test under Testbed-II setting with SPEC SFS workload.

CROCUS design tries as much as possible to provide an up-to-date GCP pool and CCP pool vector information for the purpose of accurate scheduling processes. However, the frequency of the update intervals is an important factor in the scheduling performance. As depicted in Fig. 16, which shows the sensitivity analysis with respect to varying the utilization vector update interval, lower vector update period (1 second - 3 seconds), results in lower scheduling performance as VOs scheduled per second are fewer. This is mainly caused by the excessive internal communication interference overheads. We clearly observe that the optimal number of VOs processed is achieved at periodic update time equal to 5 seconds. Finally, we observe that increasing the periodic update time to higher than 5 seconds results in a decreased number of VOs scheduled per second. This result shows that choosing the periodic update time value is vital as far as scheduling performance in CROCUS is concerned. We believe that the utilization vector update period highly depends on the workload pattern.

One of the most important functionalities of CROCUS is to identify free slots with minimal overhead. Crocus achieves this through the use of the readily available up-to-date GCP and CCP vectors. An evaluation of how the system changes the scheduling distribution with varying workload patterns on each node is therefore important. Fig. 17 captures this important distribution phenomenon. In order to conduct a detailed analysis of the VO distribution pattern of Crocus-LAFS in different contention cases, we create two scenarios. Fig. 17a shows the case when all GPU equipped nodes are equally congested, whereas, Fig. 17b depicts a scenario where GPU nodes are overloaded one at a time after every 40 seconds, in a round-robin manner. The OSS-1 to OSS-7 are CPU computation pool nodes whereas, OSS-6 and OSS-7 both have GPUs hence, they are part of GPU computation pool as well. We create contention by running SRAD application [31] for 5 seconds after every 40 seconds interval. Figs. 17a & 17b both show the highly balanced VO distribution pattern,



(a) Overloading both OSS-6 and 7 GPU nodes



(b) Overloading OSS-6 and OSS-7 in Round-robin fashion

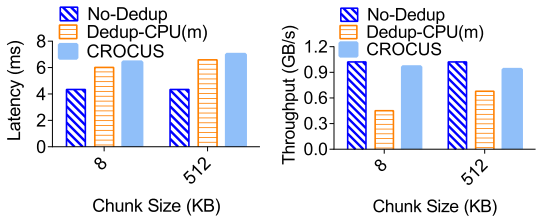
Fig. 17. Virtual object distribution pattern analysis with varying loads on Testbed-II using SPEC SFS workload.

thereby avoiding additional contention on already contented nodes. And when required, LAFS also distributes VOs to CPU nodes as shown in Fig. 17b. In particular, in Fig. 17a, we observe OSS-[6-7] shows the same trend of VO distribution, however, in Fig. 17b, we see OSS-7 drops at 40 seconds, but OSS-6 does not drop because OSS-7 is only overloaded in this experiment.

5.4 Network Overhead Analysis

In this section, we analyze the impact on performance caused by offloading tasks onto neighboring nodes. The two major overheads of CROCUS on top of the existing works are (i) memory overhead caused by buffering incoming objects before they are scheduled for deduplication processes, and (ii) network overhead, which is a result of offloading VOs to neighbor nodes. Both overheads add extra I/O time in the deduplication process. To evaluate the effects of such overheads on CROCUS perfomance, we use Testbed-II of Table 1 with the SPEC SFS database workload.

Fig. 18 shows the results of this experiment. First of all, the results in Fig. 18a, shows that enabling the multi-threaded CPU based deduplication, Dedup(m), incurs additional latency per I/O as compared to the baseline with no deduplication. This is due to the additional deduplication I/Os introduced on the I/O path. We further observe that latency is already dominated by software overhead such as fingerprinting on the baseline. The introduction of high speed networks such as Cisco Nexus technology [32], which offers cluster network speeds of up to 100 Gbps, does not eliminate this performance overhead. We also observe our approach slightly increases the latency while improving the performance overhead of the baseline in Fig. 18. However, the huge trade-off between slightly degraded I/O latency and high throughput performance shows that the network overhead of CROCUS is



(a) Latency (b) Throughput

Fig. 18. An Evaluation of CROCUS-LAFS latency versus bandwidth on Testbed-II.

insignificant as compared to the throughput gain that comes with the CROCUS implementation.

6 Related Works

Many works have been done to employ cluster-wide deduplication in distributed storage systems [11], [19], [20], [21], [23], [24], [26], [33], [34], [35]. However, many of them do not adhere to our target shared nothing architecture and none of them investigated the possibility of utilizing neighbor compute resources to accelerate the overheads of deduplication. Our work differs from the prior works in the sense that the proposed framework orchestrates the use of all cluster compute resources (CPU/GPU) and also adheres to the shared nothing design architecture. Furthermore, CROCUS does the offloading of dedup computations in a load-aware fashion, by first taking advantage of powerful GPU accelerators (if available), then distributes the fingerprinting load on the rest of the CPU resources if GPU is overloaded.

Recently, software based data deduplication products such as Intel's ISA-L [36], [37] has been introduced in the storage industry to accelerate the CPU based data reduction performance. However, CPU based data reduction methods in large scale data clusters requires an inhibitive amount of CPUs to achieve high data reduction throughput. Previous works [27], proved that, saturating a high end Suwon201622-core CPU with optimized data reduction processes using high performance Intel's ISA-L hashing, can only result in very low local data reduction throughput. Worse off, this data reduction throughput can further decrease when applied on a cluster-wide scale.

Recently, hardware accelerators such as GPUs and FPGAs have been adopted to mitigate the overheads from compute-intensive components of storage such as compression, encryption and hashing [18], [27], [38]. StoreGPU [18], introduces a general framework for hashing to provide enhanced parallelism to storage systems. However, it follows the pipeline approach for host/device communications, in contrary to the concept of stream overlapping, thus inhibiting maximum deduplication performance. Shredder [38] uses GPU for chunking and hashing in incremental deduplication-enabled storage. It improves the performance by using a stream based approach, which enables to overlap multiple data transfers and hashing computations. Shredder [38] has a limited deduplication performance improvement, because its orchestration is not cluster-wide.

One of the recent works for accelerating data deduplication in storage is CIDR [27] and [39], which uses FPGA

hardware accelerators to speed up the compute-intensive data reduction operations (deduplication + compression). However, our work differs from [27] and [39] due to their centralized metadata management which does not conform to our targeted shared-nothing storage architecture. Another drawback of [27] and [39] is that they perform localized deduplication on a single SSD array node, which does not achieve maximum deduplication opportunities across the cluster. However, CROCUS achieves this deduplication improvement by not only utilizing hardware accelerators but also considers the software based CPU acceleration with all possible compute resources across the storage cluster.

Several works such as [40], [41], [42], [43], [44] developed optimal resource scheduling in VM placement and cloud environments. However, the majority of resource scheduling for VM placement is controlled centrally, which violates the design principles of our target architecture. In [45], [46], all servers are assumed to have the same computational capabilities, whereas, our target architecture considers resource heterogeneity in the hybrid cluster environment. In [47], a novel resource provisioning approach was studied for shared nothing storage systems, using queuing network for analysis of both service performance and cost estimation of storage nodes. However, it uses a separate set of dedicated metadata servers and storage servers. In our case, we minimize the storage costs by embedding metadata management in each SSD tier node. Each SSD tier node handles the scheduling functionality. Kubernetes [43] and Slurm [44], dynamically construct mortal containers, that are temporarily created to allocate resources that runs a specific set of application workloads. As soon as the application completes its execution, these containers are terminated. This creates some problems when this approach is used to orchestrate resources in deduplication related frameworks. i) The creation of containerized resources whenever deduplication related processes are launched creates additional bottleneck. ii) Furthermore, when these containers are created on neighboring nodes, the batching on local node plus the creation of the containers at the neighbor nodes will add more overheads, thus degrading the deduplication performance.

However, none of the existing studies target dynamic and optimal resource scheduling in cluster-scale inline data deduplication. We propose to build an inline deduplication framework using compute resource orchestrations, which not only speed-up the system performance but also fully utilize the compute resources.

7 CONCLUSION

This paper presents CROCUS, a high performance cluster-wide deduplication approach for multi-tier shared-nothing storage systems. CROCUS improves the inline deduplication performance at the SSD tier with small chunks while achieving high disk space savings. Specifically, our compute-resource orchestrations exploits the available GPU resources on the SSD tier to accelerate the compute-intensive deduplication operations such as chunking and fingerprinting. Furthermore, considering the GPU heterogeneity on the SSD tier and fluctuating load on each SSD node, we design an opportunistic load-aware fingerprint scheduling algorithm, LAFS, which dynamically distributes the deduplication operations among GPU and CPU nodes. To maximize the throughput of GPU nodes, we proposed optimizations for both intra- and inter-node communications using the multi-stream based communication model. We implemented CROCUS in Ceph. The experimental results confirm the effectiveness of CROCUS and LAFS in both homogeneous and heterogeneous cluster configurations.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (Ministry of Science and ICT) under Grant NRF-2018R1A1A1A05079398.

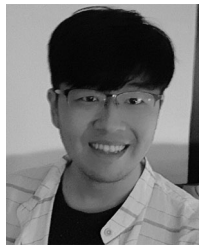
REFERENCES

[1] T. Economist, "The data deluge: Five years on," 2016. [Online]. Available: https://www.slideshare.net/economistintelligenceunit/the-data-deluge-five-years-on
[2] H. Zou, Y. Yu, W. Tang, and H. M. Chen, "Improving I/O performance with adaptive data compression for big data applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2014, pp. 1228–1237.
[3] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, 2006, pp. 307–320.
[4] Gluster, 2019. [Online]. Available: http://www.gluster.org
[5] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in HPC storage systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.
[6] W. Xia *et al.*,"A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.
[7] R. Kaur, I. Chana, and J. Bhattacharya, "Data deduplication techniques for efficient cloud storage management: A systematic review," *J. Supercomput.*, vol. 74, no. 5, pp. 2035–2085, May 2018.
[8] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 11:1–11:30, Jun. 2014.
[9] S. Lee, S. Cho, H. Kim, and Y. Won, "Performance analysis of SSD/HDD hybrid storage manager," in *Proc. 16th North-East Asia Symp. Nano Inf. Technol. Rel.*, 2011, pp. 136–139.
[10] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 20–20.
[11] A. Khan, C. Lee, P. Hamandawana, S. Park, and Y. Kim, "A robust fault-tolerant and scalable cluster-wide deduplication for shared-nothing storage systems," in *Proc. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2018, pp. 1–7.
[12] M. Oh *et al.*, "Design of global data deduplication for a scale-out distributed storage system," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 1063–1073.
[13] R. Zhu, L.-H. Qin, J.-L. Zhou, and H. Zheng, "Using multi-threads to hide deduplication I/O latency with low synchronization overhead," *J. Central South Univ.*, vol. 20, no. 6, pp. 1582–1591, 2013.
[14] W. Xia *et al.*, "FastCDC: A fast and efficient content-defined chunking approach for data deduplication," in *Proc. USENIX Annu. Tech. Conf.*, 2016, pp. 101–114.
[15] D.-Y. Lee, K. Jeong, S.-H. Han, J.-S. Kim, J.-Y. Hwang, and S. Cho, "Understanding write behaviors of storage backends in ceph object store," in *Proc. IEEE Int. Conf. Massive Storage Syst. Technol.*, vol. 10, 2017.
[16] Forbes, "GPUs in the cloud, the largest providers are investing heavily today," 2017. [Online]. Available: https://www.forbes.com/sites/benjaminboxer/2017/04/12/gpus-in-the-cloud-the-largest-providers-are-investing-heavily-today
[17] K. Jang, S. Han, S. Han, S. B. Moon, and K. Park, "SSLShader: Cheap SSL acceleration with commodity processors," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 1–14.

[18] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, "StoreGPU: Exploiting graphics processing units to accelerate distributed storage systems," in *Proc. 17th Int. Symp. High Perform. Distrib. Comput.*, 2008, pp. 165–174.
[19] C. Dubnicki *et al.*, "HYDRAstor: A scalable secondary storage," in *Proc. 7th USENIX Conf. File Storage Technol.*, 2009, pp. 197–210.
[20] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in *Proc. 13th Int. Middleware Conf.*, 2012, pp. 354–373.
[21] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proc. IEEE Int. Symp. Conf. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2009, pp. 1–9.
[22] D. Frey, A.-M. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 1–17.
[23] A. T. Clements *et al.*, "Decentralized deduplication in SAN cluster file systems," in *Proc. USENIX Annu. Tech. Conf.*, 2009, pp. 101–114.
[24] J. Kaiser, D. Meister, A. Brinkmann, and S. Effert, "Design of an exact data deduplication cluster," in *Proc. 28th IEEE Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–12.
[25] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 269–282.
[26] W. Xia, H. Jiang, D. Feng, and Y. Hua, "SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput," in *Proc. USENIX Annu. Tech. Conf.*, 2011, pp. 26–30.
[27] M. Ajdari, P. Park, J. Kim, D. Kwon, and J. Kim, "CIDR: A cost-effective in-line data reduction system for terabit-per-second scale SSD arrays," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2019, pp. 28–41.
[28] Z. Han *et al.*, "DEC: An efficient deduplication-enhanced compression approach," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst.*, 2016, pp. 519–526.
[29] SPEC, "SPEC SFS 2014," 2017. [Online]. Available: https://www.spec.org/sfs2014/
[30] Jens Axboe, "Flexible I/O Tester," 2017. [Online]. Available: https://github.com/axboe/fio
[31] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.
[32] Cisco, "Cisco nexus 7700 F3-series 12-Port 100 Gigabit ethernet module data sheet," 2019. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/data_sheet_c78-728423.html
[33] C. Lin, Q. Cao, J. Huang, J. Yao, X. Li, and C. Xie, "HPDV: A highly parallel deduplication cluster for virtual machine images," in *Proc. 18th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2018, pp. 472–481.
[34] W. Xia, D. Feng, H. Jiang, Y. Zhang, V. I. Chang, and X. Zou, "Accelerating content-defined-chunking based data deduplication by exploiting parallelism," *Future Gener. Comput. Syst.*, vol. 98, pp. 406–418, 2019.
[35] J. Ma and C. Park, "Parallelizing inline data reduction operations for primary storage systems," in *Proc. Parallel Comput. Technol.*, 2017, pp. 301–307.
[36] Intel, "Reposity of Intel storage acceleration library," 2017. [Online]. Available: https://https://github.com/01org/isa-l
[37] Intel®, "Intel® storage acceleration library (open source version)," 2019. [Online]. Available: https://01.org/intel%C2%AE-storage-acceleration-library-open-source-version
[38] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: GPU-accelerated incremental storage and computation," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 1–14.
[39] M. Ajdari, P. Park, D. Kwon, J. Kim, and J. Kim, "A scalable HW-based inline deduplication for SSD arrays," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 47–50, Jan. 2018.
[40] M. Mechtri, M. Hadji, and D. Zeghlache, "Exact and heuristic resource mapping algorithms for distributed and hybrid clouds," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 681–696, Oct.-Dec. 2017.
[41] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for MapReduce in a cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1265–1279, May 2015.
[42] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 14–28, Jan.-Mar. 2014.
[43] Kubernetes, "Production-grade container orchestration," 2019. [Online]. Available: https://kubernetes.io/
[44] SchedMD, "Slurm worload manager," 2019. [Online]. Available: https://slurm.schedmd.com/documentation.html
[45] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 370–377.
[46] X. Zhou and P. Lama, "Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 78–86, Jan. 2012.
[47] J. Zhou, J. Fan, and J. Jia, "A cost-efficient resource provisioning algorithm for DHT-based cloud storage systems," *Concurrency Comput. Pract. Experience*, vol. 28, no. 18, pp. 4485–4506, 2016.

Prince Hamandawana received the BSc honors degree in computer science from the National University of Science and Technology (NUST), Bulawayo, Zimbabwe. He is working toward the MS leading to PhD integrated program degree at Ajou University, Suwon, South Korea. He had the privilege to work for Econet Wireless, 2008-2011, and Liquid Telecom, 2011-2016. Currently, he is a member of the Database and Dependable Computing (DBDC) Lab, Ajou University with the Department of Computer Engineering. His research interests include distributed and parallel storage systems, and GPU assisted cluster-wide data deduplication.

Awais Khan (Member, IEEE) received the BS degree in bioinformatics from Mohammad Ali Jinnah University, Islamabad, Pakistan. He is working toward the MS leading to PhD integrated program degree with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. He worked in Digital Research Laboratories as software engineer from 2012 to 2015. Currently, he is a member in Laboratory for Advanced System Software in the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. His research interests include cloud computing, cluster-scale deduplication, parallel and distributed file systems.



Chang-Gyu Lee received the BS degree in computer science from Ajou University, Suwon, South Korea. He is working toward the MS leading to PhD integrated program degree in the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. Currently, he is a member in Laboratory for Advanced System Software with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. His research interests include operating systems, file and storage systems, key-value stores, and parallel and distributed systems.

Sungyong Park received the BS degree in computer science from Sogang University, Seoul, South Korea, and both the MS and PhD degrees in computer science from Syracuse University, Syracuse, New York. He is currently a professor with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. From 1987 to 1992, he worked for LG Electronics, South Korea, as a research engineer. From 1998 to 1999, he was a research scientist at Telcordia Technologies (formerly Bellcore), where he developed network management software for optical switches. His research interests include cloud computing and systems, virtualization technologies, high performance I/O and storage systems, and embedded system software.

Youngjae Kim (Member, IEEE) received the BS degree in computer science from Sogang University, South Korea, in 2001, the MS degree in computer science from KAIST, in 2003, and the PhD degree in computer science and engineering from Pennsylvania State University, University Park, Pennsylvania, in 2009. He is currently an associate professor with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. Before joining Sogang University, Seoul, South Korea, he was a R&D staff scientist with the US Department of Energy's Oak Ridge National Laboratory (2009-2015) and as an assistant professor at Ajou University, Suwon, South Korea (2015-2016). His research interests include operating systems, file and storage systems, parallel and distributed systems, computer systems security, and performance evaluation.