IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 17, 2019, accepted May 10, 2019, date of publication May 20, 2019, date of current version June 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2917841

iStore: Towards the Optimization of Federation File Systems

AWAIS KHAN[1], (Member, IEEE), MUHAMMAD ATTIQUE[2], (Member, IEEE), AND YOUNGJAE KIM[1]

[1]Department of Computer Science and Engineering, Sogang University, Seoul 04107, South Korea
[2]Department of Software, Sejong University, Seoul 05006, South Korea

Corresponding author: Youngjae Kim (youkim@sogang.ac.kr)

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

ABSTRACT With the growing volumes of data, many organizations are deploying geo-distributed edge servers and building federations atop of edge servers to improve data sharing, effective collaborations, and analytics. Multiple federation file systems are designed to satisfy such needs, but due to application-specific architectures, these federations neglect some important features that can improve the overall federation performance. In this paper, we address the important challenges of federated file systems, in particular, global namespace, optimal data placement and analysis, efficient data migration across edge servers, and metadata optimizations. To further investigate these challenges, we prototyped the federation file system *iStore* to emulate the federation and showed the significance of the afore-mentioned key challenges in the federation. The iStore provides unified global namespace atop of geo-distributed edge servers with a generic job and resource-aware data storage and placement algorithm (JRAP), which minimizes the job execution time by considering resources at each edge server. Furthermore, to enable effective data migration, we employed direct channel file layout-aware data transfer and designed a batch-based metadata scheme for federations to reduce the metadata contention with increasing clients. We evaluated the efficacy of various big data applications from data generation to storage and analysis using the iStore on real testbed and simulation.

INDEX TERMS Big data storage and HPC, geo-distributed edge computing, cluster storage and analysis.

I. INTRODUCTION

Data volume from edge devices such as sensors or mobile devices is increasing at an explosive rate, easily surpassing the scale of zettabytes [1], [2]. A single weather company can generate more than 20 terabytes of data per day alone in order to store temperature readings, wind speeds, barometric pressures and satellite images from across the globe [3]. This untamed growth of data volume poses serious challenges to Big Data applications, requiring massive scale analytical systems [4]. The current approach to handling such challenges is to use combination of core cloud services for data storing, data visualization, text-based search and map-reduce services from cloud providers. Although powerful and effective, such core cloud services are known to be not amenable to the edge computing paradigm where migration of large volume of raw data to the cloud core is considered prohibitive. Internet connectivity may be unstable, network bandwidth small and the usage cost too high.

On the contrary, small or medium-sized data centers near the edge, which we refer to as *Edge Servers*, comprising multiple machines connected via high-speed network are to guarantee high data availability and accessibility to the users. Such edge servers are typically limited in resource, lacking large scale storage space and computational capacity. We argue that these edge servers can act as a building block for a federation by combining multiple edge data centers distributed over the network in order to enhance the edge computing capability.

On the top of these geo-distributed edge servers, several scientific, research and academic communities are building federations to enable data sharing and better analytical collaborations [5]. These communities are taking advantage by sharing and analyzing data across different realms for discovering valuable products and services [5], [6]. For example, scientists and their collaborators using the Department of Energy's computational facilities typically have access to additional resources at multiple facilities and/or universities. They use these resources to analyze data generated from experimental facilities or simulation on supercomputers and

The associate editor coordinating the review of this manuscript and approving it for publication was Yuedong Xu.

to validate their results, both of which require moving the data between geographically dispersed organizations. Some examples of large collaborations include: Oak Ridge Leadership Computing Facility (OLCF) petascale simulation, which needs nuclear interaction datasets processed at National Energy Research Scientific Computing (NERSC) [6] and the Argonne Leadership Computing Facility (ALCF), which runs a climate simulation and validates the simulation results with climate observation datasets at Oak Ridge National Laboratory (ORNL) edge server [6].

The federation and grid-based file systems on top of geo-distributed edge servers are proposed to expedite such data sharing and collaborations demands [7]–[11]. However, there are several unmet challenges in existing federations file systems. The current notion of federation tends to assume homogeneous architectures, i.e., all edges across the network are required to have uniform specifications such as identical file systems, storage bandwidth, compute power and network connectivity. Existing studies in this space include Xtreem file system (XtreemFS) [8], Grid Data Farm (GFarm) [7], and HDFS [12]. However, these file systems allow the aggregation of only the identical file system. For example, the GFarm file system can aggregate other edge servers that are formatted with the same GFarm file system [7]. Such narrow requirements result in underutilizing the CPU, memory and storage resources. Besides identical file system aggregation, determining the efficient data placement in such resource heterogenic federations is also challenging. Apparently, it may sound simple: determine the data generator's location, and migrate data to the nearest edge server but sometimes, selecting nearest edge server or nearest neighbor is never an effective decision.

This simple approach ignores the following major factors. First, it does not consider the time to store the data, which is dependent on the availability of resources such as storage bandwidth and network connectivity between the edge server and data generator. Second, it does not guarantee the effective utilization of resources across the federated geo-distributed edge servers. For example, the edge server equipped with high storage bandwidth may stay idle because it is not adjacent to data generator. Third, it is not smart enough to consider the post-storage processing such as analysis of the stored data that is among the primary demands of big data. Fourth, it only uses the single edge server for job completion. These requirements raise the need of federated namespace that can facilitate the efficient data placement and analysis in the federated geo-distributed edge servers. Another challenge in federation is data migration across the edge servers. When an analysis is requested on datasets stored on two different edge servers, then datasets need to be aggregated on single edge server to run analysis. Such frequent data migration operations through federation namespace elevates the federation performance overhead. The direct channel data migration can be employed [6].

In this paper, we prototyped the federated file system iStore in order to address the significant challenges of federation file systems. iStore provides global namespace which can unify geo-distributed edge servers. iStore provides optimal data placement and analysis in the federation by considering the resource configurations of federated geo-distributed edge servers.

This paper makes the following specific contributions:

- We prototype iStore, a federation file system that provides a global unified namespace by merging different geo-distributed edge servers connected via high-speed networks such as terabits network infrastructure in DOE's ESnet [6], [13]. The aggregate storage namespace (*ASN*) is responsible to enable this virtual unification of different mounted edge servers into a single federation.
- We propose a generic job and resource-aware data storage and placement algorithm (*JRAP*), which computes optimal edge server with minimal job execution time (JET) for the job requests received via federation namespace. JRAP takes into account resource availability at each edge server participating in the federation and estimate JET at runtime to map the request to the best edge server in the federation.
- We equip iStore with a batch-based metadata approach to minimize metadata overhead and direct channel data migration (*DCM*) technique to improve data transfer in federated edge servers.
- To evaluate iStore, we build two federations, i) homogeneous resource configurations and ii) heterogeneous resource configurations at the edge servers. We execute various big data applications to show the performance of iStore on a real testbed. We also evaluate JRAP algorithm by simulation and on the real testbed to show the efficacy of the efficient placement and analysis.

II. RELATED WORK

There are few federation file systems that are developed targeting specific requirements, applications and particular areas. The parallel and distributed file systems such as Ceph [14], Gluster [15] and Lustre [16] are designed for a single-site installations, i.e. only for a single cluster. Grid Data Farm (GFarm) [7], Xtreem File System (XtreemFS) [8] and HDFS [12] are specifically designed for petascale storage and computing. SPANStore [17] introduces a geo-replicated storage service that focuses on the cost-effectivenss and delivering the key-value store service. GBFS [18] offers aggregate storage over wide-area network via grouping file systems. However, none of them are intended to provide the aggregated and unified view of file systems dispersed across data centers. Another study includes FedFS [10] and iRods [9], an object-oriented rule-based storage system which provides virtual data abstraction along with the workflow automation. These systems are limited to remote memory communications, and cannot aggregate all resources of the remote data center.

The majority of federation file systems as mentioned in [2] practice random placement, whereas random placement

is always not an efficient solution and such techniques incur both additional performance overhead and data migration cost. Additionally, implementing a flexible placement is quite complex, as it requires complete knowledge and design insight of file system under consideration. Several studies have shown interest in data-placement and data-migration approaches. IFogStor [19] implements a resource-aware placement methodology for IoT data placement. The data-migration based approaches [20]–[22] focus on migrating data with minimizing cost, whereas data-placement approaches [23]–[26] minimize the data movement instead of the data movement cost. Cho *et al.* proposed a migration-based method Pandora [20] a cost-aware model that transfers bulk data into the cloud, via both Internet and physical shipment of data. Other related studies investigating optimal placement problems have been addressed in [6], [27]. Yuan *et al.* [24] studied a data placement-based approach for scientific workflows. Few studies have shown importance of data locality and placement in Hadoop [28], [29]. Agarwal *et al.* [25] propose an automated data placement mechanism Volley for geo-distributed cloud services.

Apart from these, there are very few studies that have conducted on scavenging existing available resources such as desktop and server machines. FreeLoader [30] and Pado [31] introduce the notion of scavenging existing resources to suffice additional capacity and analytical demands. Major limitations of using NFS mount at the edge servers are the load-distribution and job-awareness. A single namespace or aggregated view of multiple NFS mount points at the single edge is not possible. Furthermore, the type of file system to be aggregated in federation is considered as the same, which we believe to be a limitation in the existing federation. Moreover, rules and policies are important in the federation where, certain users and groups are assigned specific privileges to use federation. iRODS [9] is the only federation providing automated workflow based on rules and policies. In order to adopt these federation file system the legacy data, application and architectural changes are required.

We envision iStore as a generic federation file system which can federate multiple different geo-distributed edge server file systems and provides optimal data placement and analysis along with direct edge server to edge server transfer. Moreover, iStore follows posix semantics and can be adopted atop of existing posix-compliant file systems without requiring any significant application or architectural change.

There are several key differences in our study as compared to existing federation file systems. First, existing optimization studies target cloud and not the federation environment. Second, most of the existing studies never spotlight the challenges of global namespace for federated geo-distributed edge servers with different file systems. Third, we consider optimal data placement and analysis based on job type, rules and available resources. Fourth, direct-channel data migration is not considered in existing federations design. Fifth, metadata-batching scheme is never explored before in federations.

III. iStore: FEDERATION FILE SYSTEM

In this section, we discuss our key design principles.

A. GOALS

1) GLOBAL NAMESPACE

The key design goal is to prototype a generic POSIX-compliant global namespace that can federate edge servers with different file systems requiring insignificant or no modifications in existing data, applications, software and underlying architectures.

2) RESOURCE AGGREGATION & DATA COUPLING

The edge servers partaking in the federation can throw in their resources to jobs running on other edge servers in order to optimize the job completion time. For example, if federation allows storage of data on multiple edge servers, then edge servers share their storage resources to improve job completion time. Similarly, analysis applications running on dedicated edge servers might require coupling multi-edge server sprayed data onto a single edge server. Such resource aggregation and data coupling concept can be considered but it needs detail monitoring of resources at each edge server.

3) JOB ALLOCATION MODEL (PLACEMENT AND ANALYSIS)

The efficient job allocation is significant in enhancing the overall performance of federation. The job allocation mainly depends on job type and availability of resources to perform desired tasks. For example, in the case of data placement request, it is necessary to consider an edge server with high storage bandwidth regardless of its CPU power. Moreover, if the job demands both storage and immediate analysis, a suitable combination of both storage and CPU power can complete the job in minimum time. We consider the job classification as an important design aspect of iStore.

4) RULES AND POLICIES IN FEDERATIONS

The rules and policies in a federation are not only meant to limit the data access rights but are used for different purposes such as data categorization and workflow automation [9]. For example, consider a federation with 10 edge servers $\{ES_1, ES_2, \ldots, ES_{10}\}$, where a workflow rule is implemented that, only $\{ES_1, ES_2\}$ can execute analytical applications because they are equipped with high computation power. To target such particular demands, we consider rules and policies in our design goal.

5) EFFECTIVE DATA MIGRATION

The data migration can affect the job completion time when data has to be aggregated or migrated across different edge servers. Considering the scenario of analysis between two isolated datasets stored in different edge servers in the federation. We need to migrate the datasets to one of the edge servers. First, we need to consider the best edge server that can complete the job in minimal execution time.

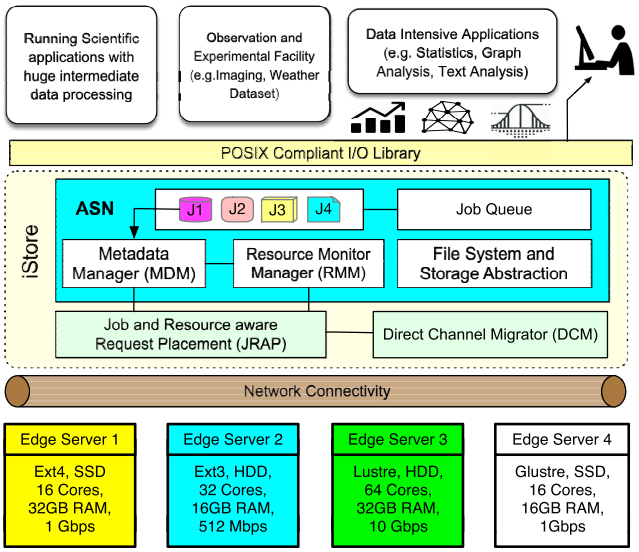

FIGURE 1. Overview of the prototype federated file system iStore.

Second, whether data has to be migrated via the federation namespace or directly between edge servers.

B. OVERVIEW

Figure 1 shows the architectural overview of iStore. The iStore prototype aggregates geo-located edge server connected via high-speed terabit network at a single mount point thereby, facilitating data sharing to enhance collaborations for effective analytics. The shim layer ASN is responsible for equipping iStore with POSIX-compliance federation and for providing file system operations interface. The file systems at the geo-distributed edge servers are mounted onto data generator via Linux NFS [32] and ASN can thus, unify all mounted edge server under a single namespace. This unification builds virtual abstraction which keeps the users unaware of the actual data location and underlying storage architectures. In order to facilitate file-to-location mapping, we designed a metadata manager which records all the federation metadata such as resource statistics at each edge server, operations and requests received via federation namespace. The metadata manager is shown in Figure 1 at ASN tier. The optimal data storage and placement component stand next to ASN and metadata manager.

As ASN catches the incoming job request, JRAP is invoked in order to execute job at the best suitable edge server with minimum job execution time. JRAP computes the optimal edge server based on job type and resource availability at each edge server in federation. JRAP takes into account the available storage capacity, storage bandwidth, computational power and network bandwidth for computing the optimal edge server. The resource monitor manager runs as a daemon on each of the edge server and provides current resource values to JRAP for optimal decision-making. Moreover, JRAP also considers the rules and policies defined in metadata manager regarding the workflow in federation. In order to further optimize the job execution time, JRAP equips iStore with parallel placement and analysis ability which requires resource aggregation from the other edge servers in the federation. Our JRAP model is generic and can be adopted by any existing federation file systems. The DCM is shown in Figure 1 alongside JRAP which enables direct channel data migration across edge servers in federation. When data needs to be migrated, JRAP triggers the DCM service on source edge server to migrate dataset to destination edge server without involving federation namespace. We claim such direct channel migration is highly efficient as compared data migration via federation namespace.

TABLE 1. Job categorizations for big data applications.

Category	Job Description
End-to-End (EE)	Requires both data storage and immediate analysis.
Placement Only (PO)	Requires only data storage and no analysis.
Analysis Only (AO)	Requires only data analysis and no storage.

TABLE 2. Summary of notations used in model formulation.

Constants	Description
$G(N, E, W)$	The graph model of a network.
W(e)	Weight of a link connecting two points.
ES	The set of edge servers.
DG	The set of data generators.
α_k	Size of slice k.
β	Number of edge servers contributing to a particular job.
Variables	**Description**
n_{ij}	Time taken to transfer one slice of data from DG_i to ES_j.
s_j	Time taken to store one slice of data by edge server c_j.
r_j	Time taken to analyze one slice of data by edge server r_j.
JET	Job execution time.

IV. JOB AND RESOURCE-AWARE REQUEST PLACEMENT

In this section, we present our job classification, system model and algorithm for optimal data storage and placement in federated geo-distributed edge servers.

A. JOB CLASSIFICATION

While big data is now in vogue, many organizations are using the private cloud federations for their applications. Each application has different job request based on their functionalities, thus it changes the patterns of data processing. Some applications require immediate analysis on generated data while others are more focused on investigating legacy data. Therefore, we divide the job requests into three categories as shown in Table 1.

In rest of the paper, we use EE, PO and AO to refer these jobs.

B. SYSTEM MODEL

For the convenience of the readers, major notations used in this paper are listed in Table 2. We consider a private network consisting of set of geo-distributed edge servers $ES = \{ES_1, ES_2, \ldots, ES_n\}$ and a set of data generators (e.g., weather sensing satellites) $DG = \{DG_1, DG_2, \ldots, DG_n\}$ that are continuously generating large volumes of data. The data generator connects to the edge

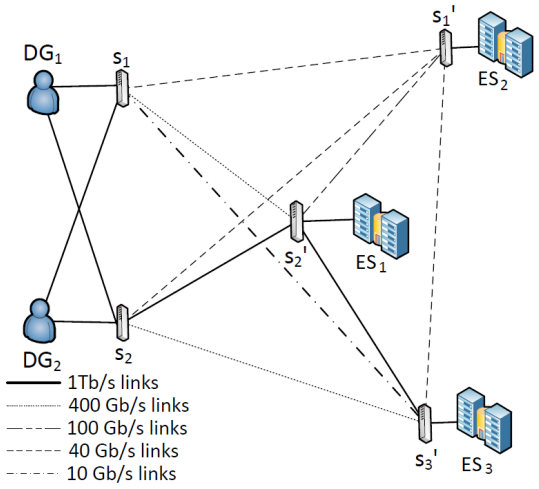

FIGURE 2. An illustration of edge servers in federation.

server via dedicated high bandwidth network with S VPN switches at the user side and S' VPN switches each collocated with an edge server.

Figure 2 provides an overview of our network design, which is modeled as an undirected weighted graph $G = (N, E, W)$, where N denotes the set of nodes $N = \{DG_i \in DG, s \in S, s' \in S', ES_j \in ES\}$, E is a set of edges that connects two distinct nodes and W(e) denotes the weight of an edge e. Weight can be the amount of time taken to transfer one slice of data from one node to another.

JRAP exploits the benefits of parallel processing and splits the data across multiple edge servers ensuring the optimal time. Splitting the data across all the edge servers in the private cloud may give a minimum job execution time however it increases the migration and metadata management overheads. Therefore, we introduced a parameter β, which is a positive real number ($\beta \in \mathbb{R}^+$) controlling the number of edge servers used for particular request. Our algorithm finds the set of β number of optimal edge servers. The workload is distributed among edge servers proportionally based on the resources of each edge server. Specifically, the edge server with powerful resources gets the maximum share of workload. Note that β is defined by the data generator based on their requirement. For example, $\beta = 1$ means user wants all the data to be stored in single edge server.

1) TRANSFER TIME

Consider the data generator DG_i wants to send the x_{ij} amount of data to edge server ES_j. Let α_k be the size of slice and p be the number of slices needed to be transferred from data generator DG_i to edge server ES_j. The time taken to transfer one slice of data from DG_i to ES_j is denoted as n_{ij}. Therefore, the transfer time $t_{tr_{ij}}$ to send x_{ij} amount of data from DG_i to ES_j is represented as $t_{tr_{ij}} = p.n_{ij}$.

$$t_{tr_{ij}} = p.n_{ij} \tag{1}$$

JRAP utilizes the parallel processing and consequently the data are simultaneously transferred to β edge servers. The overall transfer time is given as:

$$t_{tr} = max(t_{tr_{ij}}, t_{tr_{ik}}, \ldots, t_{tr_{i\beta}}) \tag{2}$$

2) STORAGE TIME

It is an important factor to be considered in choosing the edge server for data placement. Let's consider s_j is the time taken to store one slice of data by edge server ES_j. The storage time to store x_{ij} amount of data by ES_j data is denoted as $t_{st_j} = p.s_j$.

The overall storage time is represented as:

$$t_{st} = max(t_{st_j}, t_{st_k}, \ldots, t_{st_\beta}) \tag{3}$$

3) ANALYSIS TIME

The server with a high computation power is more likely to be chosen for data analysis. Let r_j be the time taken to analyze one slice of data and p be the number of slices needed to be analyzed. The analysis time to analyze x_{ij} amount of data by ES_j is represented as $t_{an_j} = p.r_j$.

The overall analysis time is represented as:

$$t_{an} = max(t_{an_j}, t_{an_k}, \ldots, t_{an_\beta}) \tag{4}$$

Note that in current scope of study, we consider simple application scenarios and analysis time is estimated based on history knowledge of the application. Therefore, analysis time of many common big data applications such as Grep (GR) and Group-by Aggregation (GAG) can be estimated based on data size and CPU performance.

C. JOB ALLOCATION MODEL

In this section, we present our approach for managing all the three categories of job requests.

1) END-TO-END PLACEMENT AND ANALYSIS (EE)

This type of job requires both data placement and immediate analysis on generated data. Therefore, we try to optimize the data storage and analysis considering the combination of data routing, data storage and analysis constraints. We use the Job Execution Time (JET) of a data processing request as a metric of selecting a set of optimal edge servers. A straightforward way to reduce JET is by deploying the high-speed network between the data generator and edge server. For example, DOE's ESnet currently supports 100 Gb/s between DOE facilities, and in future deployments will most likely support 400 Gb/s followed by 1 Tb/s [13]. However, these network improvements only increase the latency of data transferring process, not end-to-end data placement and analysis time. Therefore, the storage bandwidth and computational power of edge server also play a vital role in achieving the minimum JET. In addition, JRAP also needs to consider the current workload and the availability of resources in the edge server. Technically, the edge server with less available resources yields high JET as compared to the edge server with maximum available resources such as storage bandwidth, storage capacity and computational power.

Assume the data generator $DG_i \in DG$ has initiated an EE request at a certain period. The total JET to be minimized

has three components: transfer time t_{tr}, storage time t_{st} and analysis time t_{an}. Recall that t_{tr} is the data transfer time from data generator to edge server, t_{st} is the data storage time taken by edge server and t_{an} is data analysis time. Putting these times together, we can get the cumulative job execution time of ES_j.

$$JET_{ES_j} = max(t_{tr_{ij}}, t_{st_j}) + t_{an_j} \tag{5}$$

Here, $max(t_{tr_{ij}}, t_{st_j})$ represents the data placement time. Since we use slice as a unit in data processing, edge server begins writing operation as soon as it receives the first slice. Therefore, to avoid time overlap we use the maximum of the transfer time and storage time.

Parallel Data Placement and Analysis: Our objective function is to determine a set of optimal edge servers that minimizes the overall job execution time. Therefore, our algorithm splits the data across multiple data centers ensuring the optimal time. Splitting the data across all the edge servers in the private cloud may give a minimum data placement time however it increases the migration and metadata management overheads. Thus, we introduced a parameter β, which controls the number of edge servers used for particular job request. Our algorithm finds the set of β number of optimal edge servers and the data is distributed among data centers proportionally based on the resources of data center. Note that β is defined by the data generator based on their requirement. The overall JET of job request is the maximum JET consumed by any edge server. We can represent it as:

$$JET = max(JET_{ES_1}, JET_{ES_2}, \ldots, JET_{ES_\beta}) \tag{6}$$

2) DATA PLACEMENT ONLY (PO)

In most big data applications, the data is produced continuously from different geographical locations and analysis is not pre-defined. We next design a model that automates the data placement only (PO) job requests by exploiting the same parallel processing.

The data placement time t_{pl} at ES_j can be formulated as: $t_{pl_j} = max(t_{tr}, t_{st})$. Since the data request is sent to each edge server simultaneously. Therefore, the maximum data placement time consumed by any edge server considered as aggregated T_{pl}. We can represent it as:

$$T_{pl} = max(t_{pl_1}, t_{pl_2}, \ldots, t_{pl_\beta}) \tag{7}$$

3) DATA ANALYSIS ONLY (AO)

For data analysis only jobs, we adopt an on-site analysis technique, whose basic idea is to perform analysis on the same edge server unless migrating data to any other edge server improves analysis time. Therefore, the data migration occurs only in two cases: (1) Data center ES_j does not have any computation power, and (2) The data from ES_j is migrated to ES_k if $t_{an_j} > (t_{mi_{jk}} + t_{an_k})$. Here t_{an_j} is analysis time on ES_j, $t_{mi_{jk}}$ is migration time from ES_j to ES_k and t_{an_k} is analysis time on ES_k. The migration time can be seen as T_{pl}, the only difference is t_{tr} depends on network bandwidth between edge servers instead of data generator to edge servers.

Algorithm 1 Job and Resource-Aware Placement Algorithm

Input: DG, list of data generator; ES, list of edge servers; x_{ij}, requested data size; β, number of edge servers used for data placement, JT, Job Type
Output: ES_{optset}, Optimal edge server set

$ES_{sort} \leftarrow sort.ES(JET_{ES_j}, JT)$ /*sort ES list by Job Execution Time*/
$ES_{candopt} \leftarrow ES_{sort_{i=1\ to\beta}}$ /*optimal set contains first β edge servers from sorted list*/
$allocate.size(ES_{candopt})$
$notifyresource_{usage}(ES_{candopt})$
$ES_{candopt}.JET = ComputeJET(ES_{candopt,JT})$
$findopt = false$
while $findopt \neq true$ **do**
  for *each* ES_j *in* $ES_{candopt}$ **do**
    if $ES.availabcap > allocate.size$ **then**
      $JET_{opt} \leftarrow JET$
      $ES_{optset} \leftarrow ES_{candopt}$
      $findopt = true$
    end
    $JET = ComputeJET(ES_{candopt_{size}})$
    $ES_{victim} \leftarrow ES_{candopt}.pop()$ /*pop ES which does not have required available capacity*/
    $ES_{candopt} \leftarrow ES_{sort}.push()$ /*push next ES in $ES_{candopt}$ from sorted list*/
    $JET_{new} = ComputeJET(ES_{candopt})$
    if $JET_{new} < JET$ **then**
      $JET_{opt} \leftarrow JET_{new}$
      $ES_{optset} \leftarrow ES_{candopt}$
      $findopt = true$
    end
  end
end
return ES_{optset}

The total analysis time T_{an} of ES_k can be formulated as:

$$T_{an_k} = t_{mi_{jk}} + t_{an_k} \tag{8}$$

In the case of on-site analysis $t_{mi} = 0$. Similar to aggregated T_{pl}, we can compute aggregated analysis time $T_{agg_{an}}$ of job request performed by n number of edge servers:

$$T_{agg_{an}} = max(T_{an_1}, T_{an_2}, \ldots, T_{an_\beta}) \tag{9}$$

D. JOB AND RESOURCE-AWARE PLACEMENT ALGORITHM

In this section, we present our job-aware algorithm (Algorithm 1) that can manage each job based on their type. To simplify the presentation, we consider EE job category in the following description and pseudocode. Our algorithm, determines the β number of optimal edge servers for each EE request given complete knowledge of data generation in both spatial and temporal domains. The decision making of

algorithm depends on the current resource availability at each edge server. At first, algorithm sorts the edge servers based on the JET_{ES_j} of each edge server and generates the candidate optimal set $ES_{candopt}$ by taking first β edge servers from the sorted list. We then distribute the data among the edge servers based on the resources of each edge server and compute the optimal JET_{opt}. However, this $ES_{candopt}$ and JET_{opt} may not remain valid if any of the edge server in $ES_{candopt}$ has less available storage capacity than the allocated data size. In such cases, the algorithm assigns the remaining data to other edge servers in candidate list and re-calculate the T_{pl}. Then, $ES_{candopt}$ is updated by discarding the edge server with less available capacity and inserting next edge server from the sorted list. We compute the JET_{new} of updated $ES_{candopt}$. Algorithm checks if $JET_{new} < JET$ then, we update our JET_{opt} and ES_{optset} set. Eventually, the algorithm terminates by returning ES_{optset} with minimum JET.

The Algorithm 1 considers the EE jobs, however, it can be adapted with minor modification for PO and AO jobs. For PO, first $ES_{candopt}$ is computed by sorting the edge servers based on t_{pl} and then we use the formula presented in equation (7) to compute the minimum data placement time. Similarly, for AO, to choose the edge server for migration, we simply sort the edge servers based on T_{an} and then, use the equation (9) to compute the $T_{agg_{an}}$.

E. EXTENSION TO DYNAMIC JRAP

Our current JRAP uses the static model which made decisions based on the information available at the time of job arrival. Our proposed technique can be easily extended for dynamic model where decision can be changed in the middle of the job execution, if needed. Consider a scenario, when JRAP receives a job job_2 at that time the most powerful server ES_1 was occupied with job_1 and therefore JRAP sends it to next best available server ES_2. After a while, the ES_1 has finished job_1 and now JRAP will shift the job_2 from ES_2 to ES_1 if it improves the overall job execution time.

The resource monitor manager (RMM) notifies the resource availability to JRAP when a job execution is completed on any of the edge servers. Consider ES_m finishes the job which increases its available resources. JRAP will check the if it can transfer any current job in progress from other server ES_n to ES_m that improves the overall JET which is computed using equation (6). The overall JET will only be improved if following condition is satisfied.

$$JET_{ES_m} + t_{mi_{nm}} < Rem.JET_{ES_n} \tag{10}$$

Here $t_{mi_{nm}}$ is the migration cost of data from ES_n to ES_m and $Rem.JET_{ES_n}$ is the remaining JET of ES_n to complete the job. We maintain a job queue in the MDM, which contains the metadata of each job, i.e., the servers participating in the job, percentage of workload share given to each edge server. A simple job progress monitor can be integrated to manage job progress of each edge server against the job by adding start and end time of job. The migration cost incurs because β is defined at the start of the job therefore data has to be migrated from ES_n to ES_m to maintain the exact β number of servers involved in the job execution. The following two points are important in dynamic decision making of JRAP. First, the edge server ES_m most likely be assigned to the job that is in initial phase of execution. Therefore, we maintain a job queue and JRAP starts examining the job from most recent because generally the jobs at the end of the queue are already near completion and may require higher migration cost. Secondly, ES_m takes over the job from the weakest server among β. Hence, JRAP just compares the last edge server ES_β with the available edge server ES_m.

Note that, the JRAP extension algorithm is suitable for end-to-end (EE) and placement-only (PO) only jobs. It is because additional migration cost on already stored data can slow down analysis job, resulting in higher JET than expected.

V. DESIGN AND IMPLEMENTATION

In this section, we describe our rationale behind the ASN design and implementation, batch-based metadata manager (MDM), data storage and placement manager (JRAP), direct channel data migrator (DCM) and finally, resource monitor manager (RMM).

A. ASN: AGGREGATE STORAGE NAMESPACE

We prototyped aggregate storage namespace (ASN) in FUSE's high-level API v2.9.4 [33], [34]. File System in Userspace (FUSE) is the most widely used framework to prototype and evaluate new approaches to file system design [33]. We studied FUSE's high-level design, internal architecture and implementation in detail. FUSE has evolved over the time and various optimizations are being added to FUSE such as *big_reads* and *big_writes*, *single* and *multi-thread mode*. To best of our knowledge, the latest FUSE library can utilize *max_writes*, *writeback-cache*, and *splicing* features [33].

We implemented all the basic file system operations in ASN such as *init, access, create, getattr, mkdir, read, readdir, write* and other essential functions. When iStore is mounted, *asn_init* is the first method to execute. The metadata manager node address is supplied to iStore at mount time as an argument. *asn_init* first checks the metadata manager service is up and running, then it initiates a request to metadata manager to provide a list of contributing edge servers in federation with their complete configurations. After receiving the response from metadata manager, *asn_init* fills its private data structure to keep the list of edge servers contributing in iStore federation. On the contrary, the *asn_destroy* initiates the unmount request and releases all the occupied resources. The metadata-oriented methods like *asn_create* and *asn_open*, all require metadata manager assistance in order to get the file location in the federation, i.e., which edge server contains the file. When data-oriented methods such as *asn_write* is invoked, only a 4KB of data is sent to the user daemon for writing but when *big_writes* optimization parameter is passed as an argument to iStore, then bigger chunks of data are sent to the user daemon. This chunk size value is configured in

FUSE configuration file with key as *max_writes*. We used *max_writes* of size 1MB in order to observe better and consistent performance throughout our implementation and evaluation. To further improve FUSE performance, we believe that host-side caching based on SSDs can be taken into account but we did not consider in iStore prototype implementation because the goal of our study is to highlight the significance of various features in the federation which previous studies did not consider in their design.

B. MDM: METADATA MANAGER

The metadata manager (MDM) in the iStore design is of vital importance. First, it interacts with almost all the iStore components. Secondly, all requests received via the iStore federation namespace require assistance from the MDM to complete the operations. Thirdly, the federation metadata statistics such as the total no. of edge servers, their default resource configurations are maintained by the MDM.

We implemented MDM using gRPC, a high performance, open-source, multi-platform, language neutral RPC framework for developing distributed applications and services [35]. MDM is defined as a gRPC service, specifying the methods that can be called remotely by ASN. gRPC uses Google protocol buffers (Protobuf) for underlying the message interchange format [35]. Protobuf is Google's open source mechanism for serializing structured data [36]. We defined standardized messaging format in Protobuf for communication among all the components. The file system metadata e.g., stat, size, date are maintained by the file system. We keep only edge server and file mapping in MDM. The important methods implemented in MDM service includes, *init_istore*, *create_md*, *rm_md*, *create_batchmd*. Here, create and remove are simple metadata operations similar to other metadata management approaches whereas, *create_batchmd* is not explored in federations before. Our batch-based scheme hold the metadata-related I/Os in memory upto a certain limit, which we call *batch_size*. All I/O requests are buffered until the *batch_size* is reached. Once the batch threshold exceeds, single query is sent to MDM with multi-valued insert thus reducing the contention which can be incurred by using one-to-one remote connections to MDM for each request. The Algorithm 2 shows the proposed batch-based metadata flush algorithm. However, there is a certain fault-tolerance issue interrelated to this batch-based scheme but we believe that can be remedied by employing persistent memory storage architectures such as Flash or PRAM.

To the storage end of MDM, we used SQLite, a lightweight, public domain C library that does not require any configuration, can easily be embedded into an application, supports databases up to two terabytes in size [37]. It is the best choice to use SQLite API in our research because of its simplicity and support for public domain status. MDM design comprises of multiple relational schemas, each responsible for holding different kind of information. We used multiple schemas in order to manage information properly and to reduce query contention on single schema. Moreover, our MDM requires minimal modifications to offer searchable federation namespace. The existing studies have shown the needs of searchable namespace in large scale storage systems [38]–[40].

Algorithm 2 Batch-Based Metadata Flush Algorithm

Input: *WIO*, new incoming Write I/O; *CB*, current batch data structure; *CT*, current time of IO request arrival
Output: $Flush_{decision}$ /*Decision to flush batch*/

```
SBS ← loadConfigs() /* batch size threshold for metadata */
BT ← loadConfigs() /* batch time threshold for metadata */
Flush_decision ← false
Time_check ← 0
CB.add(WIO)
if CB.size() >= SBS.size() then
    Flush_decision ← true /*if size is equal to batch size*/
end
else
    Time_check ← CT − CB.time()
    if Time_check >= BT.time() then
        Flush_decision ← true /*if time is equal to batch time */
    end
end
return Flush_decision
```

C. JRAP: DATA STORAGE AND PLACEMENT MANAGER

The JRAP manager is responsible to control and balance the data flow in federation. JRAP manager depends on few parameters for computing the optimal edge server at runtime. The parameter includes job type, available capacity, storage, computation and network bandwidth. The latest resource values are provided by the resource monitor manager running on each edge server in federation. Once all values are provided to JRAP as input, JRAP first filters the edge server list based on job type and available capacity. Then JRAP computes the job execution time (JET) on the filtered edge server list. The JRAP output is optimal edge server with minimum job execution time. The output is sent to metadata manager to store the file and its location. The JRAP manager is implemented as part of ASN, where all data related-operation *create*, *write* and *read* consult JRAP to provide edge server, which can complete job in minimum time. The JRAP manager implements the Algorithm 1. The details of algorithm and JRAP decision-making is presented in Section IV.

D. DCM: DIRECT CHANNEL MIGRATOR

The direct channel migrator (DCM) is accountable for data migration job across edge servers in the federation. The principal goal to develop direct channel data migration is to

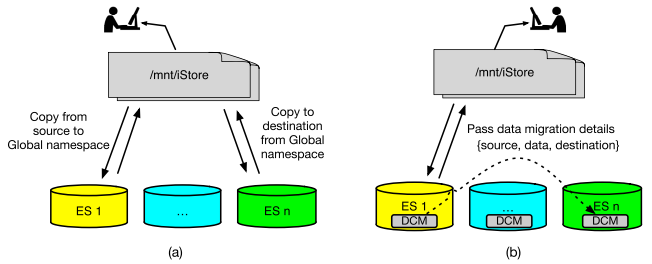

FIGURE 3. An illustration of data transfer round trip. (a) Data migration without DCM. (b) Data migration with DCM.

optimize the task completion time. For example, in order to execute analysis on an edge server with high computation power, the data first needs to be migrated to that edge server as shown in Figure 3 (a). Such cases are likely when data is stored recognizing placement only job. As, placement only job takes into account storage capacity and storage bandwidth, ignoring computation power. In such data migration scenarios, data is read through the ASN layer, comprising of FUSE-based implementation which slowers the data migration and then, again stored on the edge server responsible to run analysis via the ASN layer. However, we propose to integrate a direct channel data migrator (DCM) in iStore to minimize such ASN layer overhead. In DCM, all edge servers are in contract to communicate and share data with each other at the backend, i.e., bypassing ASN layer to expedite data migration as shown in Figure 3 (b).

We implemented DCM using C/C++ and integrated it with data storage and placement (JRAP) manager. Whenever JRAP detects any sort of data migration requirement, it triggers DCM service on source with a complete request format. A simple request includes, file location, file offset, transfer size, destination. We designed DCM based on the idea of layout-aware data transfer proposed in the study [6]. DCM is multi-threaded implementation, It consists of two major components, i.e., *dcm_src* is the source edge server which needs to transfer data whereas, *dcm_dest* is the destination edge server on which data is going to be stored or analyzed. DCM blocks all the update operations on data being migrated.

E. RMM: RESOURCE MONITOR MANAGER

The resource monitor manager is implemented for the purpose of collecting the snapshot of available resources. JRAP is highly dependent on RMM for providing the resource availabilities. RMM collects the snapshot in two modes, i) Time interval and ii) On-Demand. The reason for using time interval is to ensure the status of edge server in Federation. RMM based on certain time-interval notifies the MDM about the active edge servers. In case, any edge server leaves the Federation due to failure, RMM notifies the MDM to exclude failed edge server from the list used by JRAP algorithm. Whereas, On-Demand mode is used on arrival of any request. RMM is invoked and latest available resources are collected.

VI. EVALUATION

We now detail our evaluation methodology for the proposed system.

A. EVALUATION SETUP

1) TESTBEDS

We evaluated iStore on two different testbeds shown in Table 3. The *Testbed I* comprises of four nodes each with disk bandwidth of 81MB/s, 16 Cores x 2.60GHz, RAM 32GB and connected via Infiniband (56 Gb/s). In order to spotlight proposed optimization features, we constructed a small private cloud federation using four desktops machines with varying resource configurations at each machine shown as *Testbed II* in Table 3. Each machine is considered as a geo-distributed edge server. The edge servers are mounted at the data generator using Linux NFS [32]. We build an Ext4 [41] federation via iStore federation file system prototype. There are two reasons to use the Ext4 file system federation. First, we have a limited small-scale testbed environment and second, it is common, stable, and has a well-documented design that facilitates performance analysis. Before every experiment, we drop system cache and re-mount the iStore. For the evaluation of JRAP for large-scale edge servers environment, we developed a iStore simulation framework written in C/C++ and performed a set of simulations. A detailed description of the iStore emulator is provided later.

2) WORKLOADS

To fairly evaluate our prototype iStore federation, we used four realistic big data applications.

TABLE 3. Testbed configurations. Testbed I is homogenous and Testbed II is heterogeneous. In table, capacity, S. Bw, and IB denote storage capacity, storage bandwidth, and Infiniband network connection.

Testbed	No. of Nodes	Capacity (GB)	Storage Bw (MB/s)	CPU (GHz)	RAM (GB)	Network (Mb/s)	Big Data Applications (MB/s)
Testbed I	4 ESs	1000	81	2.60 x 16 Cores	32	IB (56 Gb/s)	
Testbed II	ES 1	120	116	2.40 x 8 Cores	8	942	WC = 160, AG = 135, GR = 105, GAG = 70
	ES 2	300	86	3.20 x 8 Cores	16	239	WC = 250, AG = 220, GR = 160, GAG=115
	ES 3	280	182	2.20 x 8 Cores	4	91	WC = 82, AG = 74, GR = 60, GAG = 46
	ES 4	100	116	1.70 x 4 Cores	8	942	WC = 64, AG = 54, GR = 48, GAG = 32

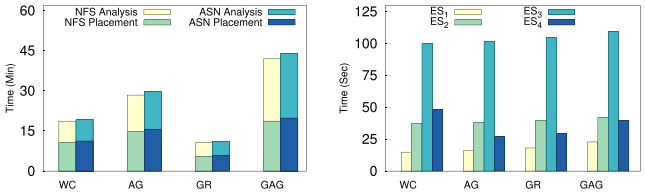

FIGURE 4. iStore aggregate storage namespace overhead with baseline NFS for various big data applications with $\beta = 1$. (a) Testbed I. (b) Testbed II.

- *Group-by Aggregation* (GAG) computes the statistical summary based on the certain group, e.g., the total sum and average hourly temperature datasets based on the city. Each line of the input file is composed of country, city, date and hourly temperature. The calculations are performed by grouping them based on the city name.
- *Aggregation* (AG) works similar to GAG, but it calculates the global statistical summary instead of taking the certain group into account, i.e., not based on city or any other group by value. We implemented AG by slightly modifying the GAG code.
- *Grep* (GR) is a string matching application that print lines containing a specific keyword from the input dataset.
- *Word-Count* (WC) counts the total bytes, characters and words, in the dataset.

The computational power of each edge server is considered to be heavily dependent on the workload type. For example, a simple application such as word count does not consume much of computational resources (i.e., CPU cycles) and takes less analysis time whereas complex applications such as image processing consume computational bandwidth of the system. In this experiment, we used weather data of different cities in the United States containing hourly readings of temperature, wind speed and humidity for past 3 years. To ensure the accuracy of results, we reiterated each of our experiment 5 times and measured the average of results. The analysis time of above mentioned applications with different data sizes were pre-collected and stored in MDM.

B. PERFORMANCE ANALYSIS

To show the overhead of iStore ASN layer on top of Linux NFS [32], we defined an end-to-end placement and analysis workflow i.e., from data generation to data analysis. At first, we examined Word-Count (WC) application. The WC is a simple application that counts the total number of words in a given dataset. We used a data set of 40GB for this application. We measured storage and computational bandwidth of NFS and set it as performance baseline. Then, we measure ASN performance for both storage and analysis. The experiment shows overhead caused by ASN on top of geo-distributed edge servers in Figure 4(a). The results encountered near baseline performance with a negligible overhead in both storage and analysis for the WC application. Aggregation (AG) being a little more complex than WC application aggregates all the data based on the certain parameter. For AG we supplied a workload size of 55GB. We observed a uniform ratio of storage and analysis overhead in Figure 4(a) for ASN and NFS.

To target compute-intensive applications, we consider GREP (GR), textual searching or string matching in big datasets based on given parameter. The GR can be high resource-intensive if the number of matching pattern increases in dataset. We analyzed GR with 20GB of workload. Our results reflected the exact traces of WC and AG near baseline performance with minimal overhead. To further investigate the overhead, we raise the compute intensity to a significantly high level by running the Group-by Aggregation (GAG) application. The GAG is an extended style of AG where all dataset is not grossed but the data set is grouped based on certain parameters provided. We adopted a workload of 70GB to evaluate GAG. The experimental results in Figure 4(a) conclude that the overhead caused by aggregate storage namespace layer is quite negligible and application complexity does not impact the overhead, however, the overhead can vary linearly with increasing workload size.

Next, we discuss the big data application runtime with respect to different edge server configurations as shown in Figure 4(b). ES_2 is more powerful in terms of computational power, as presented in Table 3. However, the overall JET is higher than ES_1 because of slower network bandwidth causing huge latency in placement and analysis. Although ES_3 is equipped with SSD and has high storage bandwidth, experimental results shows that ES_3 performs the worst due poor computational power and network bandwidth. These result can also be validated from Table 4, note that ES_2 is optimal edge server while considering only CP whereas ES_3 is

TABLE 4. Job execution time and optimal data placement with various combination of resources for each big data application with $\beta = 1$. The results in parenthesis shows the second optimal edge server.

Workloads	Decision	iStore$_{JRAP}$	SB, NB	SB, CP	CP, NB	SB	CP	NB
WC	Optimal	ES_1	$ES_1(ES_4)$	ES_2	ES_1	ES_3	ES_2	$ES_1(ES_4)$
	JET (Sec)	23	23(47)	96	23	222	96	23(47)
AG	Optimal	ES_1	$ES_1(ES_4)$	ES_2	ES_1	ES_3	ES_2	$ES_1(ES_4)$
	JET (Sec)	24	24(53)	97	24	224	97	24(53)
GR	Optimal	ES_1	$ES_1(ES_4)$	ES_2	ES_1	ES_3	ES_2	$ES_1(ES_4)$
	JET (Sec)	26	26(58)	98	26	229	98	26(58)
GAG	Optimal	ES_1	$ES_1(ES_4$	ES_2	ES_1	ES_3	ES_2	$ES_1(ES_4)$
	JET (Sec)	30	30(78)	100	30	245	100	30(78)

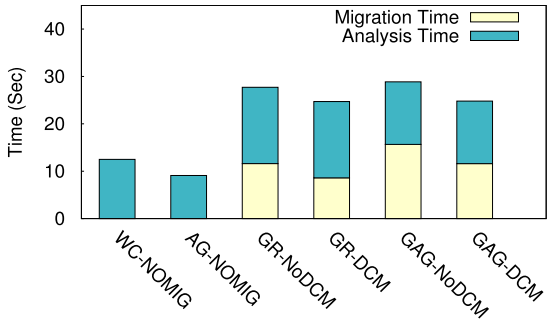

FIGURE 5. Analysis only jobs in workflow defined federation evaluated on *Testbed-II* with and without DCM.

optimal in case of SB, but both these edge servers have high JET than ES_1.

ES_1 and ES_4 share same storage and network bandwidth, however because of less computational resource availability ES_4 showed quite disturbing JET, when it is compared to ES_1. As we discussed earlier, there is light-weight daemon running on each edge server monitoring latest job execution status to UniMD. These statistics about edge servers are then gathered by uni-data placement at runtime. At that particular instance of time, resource utilization at ES_4 was at peak and uni-data placement agent dis-regarded ES_4 to be more efficient than ES_1 and gives higher JET than ES_1. Table 4, also reflects the same results, notice that ES_1 and ES_4 are optimal while considering (SB, NB) and NB only because both have same resources. However, due to utilization of computational power ES_1 yields the minimum JET and therefore it is the optimal ES while considering all resources.

C. DIRECT CHANNEL DATA MIGRATION ANALYSIS

Figure 5 shows the analysis only (AO) job evaluation with efficient direct channel migration in federation with a certain workflow rule defined. This experiment is conducted on 4 nodes of *Testbed II*. To emulate the federation rules, we defined a workflow rule in MDM that, the edge servers ES_1 and ES_2 can only run analytical jobs in federation because of high computation power. We first stored 1 GB of data on each edge server including analytical edge servers, i.e., ES_1 and ES_2. We conducted this experiment to show the effectiveness of DCM in federation. In Figure 5(c) WC and AG are executed without any migration because they were already stored on ES_1 and ES_2, whereas, datasets stored on ES_3 and ES_4 has to be migrated to any of the analytical edge server. The JRAP manager determines the destination edge servers to migrate datasets to ES_1 and ES_2 based on job type and resources. The NoDCM in this experiement refers to no direct channel data migration and requires data to be transferred via federation namespace. The datasets first need to be transfered to the iStore federation client and then transferred to a specific edge server where analysis will run. The experimental results depict that, even in small scale federation, DCM can impact the performance.

TABLE 5. Batch-based metadata evaluation using MDTest [42] evaluated on *Testbed-I*. Batch Size refers to number of I/Os aggregated in one batch.

File Ops	NFS	Batch Size				
		0%	25%	50%	75%	100%
Create	5107	13	325	769	1100	2102
Stat	765754	15083	16843	16575	27264	27264
Remove	9982	5921	6048	6571	6711	7285

D. BATCH-BASED METADATA EVALUATION

In this section, we present evaluation of proposed batch-based metadata scheme in iStore prototype federation. This experiment is conducted on *Testbed I*. We used MDTest [42] to evaluate batch-based metadata scheme. We first run the experiment on Linux NFS and then, evaluate iStore metadata performance considering Linux NFS as baseline. The evaluation results obtained from MDTest are shown in Table 5. Each result shows the mean value of file operation per second for total 5 iterations of 16,384 files. The Batch Size in table refers to number of entries in the batch. The 0% shows no batch scheme whereas, 25% denotes percentage of I/Os aggregated in one batch and then, send single request to metadata manager in form of multi-valued insert query. Similarly, 100% shows, that all requests were batched and single multi-valued insert query is sent to metadata manager. Table 5 shows high improvement with batch-based scheme towards the increasing size of batch. The metadata manager could only provide almost half of Linux NFS metadata performance, when we used single batch for all the I/Os. The reason of this degraded performance in batch size 0% and 100% as compared to linux NFS is two folds. First, for every file create operation, FUSE invokes five operations serially, *getattr*, *lookup*, *create*, *write* and *flush*. Second, user and kernel space context switching overhead cannot be ignored.

VII. JRAP FOR EDGE SERVERS CLUSTER

For the evaluation of JRAP, we conduct experiments on both the simulation and real testbed environments. First, we present the experimental results on the simulation environment where we consider a private cluster comprised of 50 edge servers. Next, we show the experimental results in real testbed where we intend to deploy a realistic cluster environment and find the optimal edge server for an end to end data placement and analysis for different applications. We investigate the importance of each decision parameter that is Storage Bandwidth (SB), Network Bandwidth (NB) and Computational Power (CP) in choosing the optimal edge server.

1) SIMULATION FRAMEWORK

We developed our JRAP solver using C/C++ language. The source codes are less than 500 lines of codes. The solver execution time is extremely short (in milliseconds). All simulation experiments are performed on a desktop PC with a Pentium 2.8GHz processor and 4GB memory. We designed private cluster federation with 50 edge servers, $Cluster_{50}$.

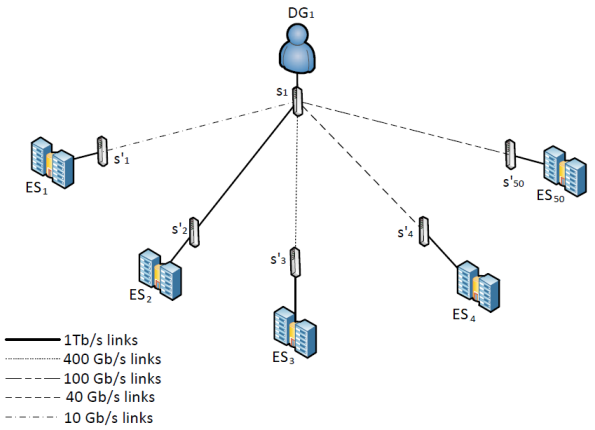

FIGURE 6. Depiction of geo-distributed network connected 50 Edge Servers.

TABLE 6. Description of federation configurations.

Parameters	Range
Available Capacity (TB)	200, 400, 600, 800, 1000
Storage Bandwidth (GB/s)	5, 10, 20, 30, 40, 50
Computational Power (GB/s)	3, 6, 12, 18, 24, 30
Network Bandwidth (Gb/s)	10, 40, 100, 400, 1000

TABLE 7. Job execution time for EE Jobs.

Decision	JRAP	(SB,NB)	(SB,CP)	(CP,NB)	(SB)	(CP)	(NB)
ES for Job_1	ES_{14}	ES_{12}	ES_{28}	ES_{33}	ES_6	ES_{24}	ES_{40}
JET (Sec)	85	191	241	133	855	233	183
ES for Job_2	ES_{33}	ES_{19}	ES_{28}	ES_{17}	ES_6	ES_{24}	ES_8
JET (Sec)	73	187	97	113	293	132	146
ES for Job_3	ES_{28}	ES_{41}	ES_{28}	ES_9	ES_6	ES_{24}	ES_{37}
JET (Sec)	61	94	61	121	123	141	90

Figure 6 shows a depiction of private cluster $Cluster_{50}$. In order to cover various realistic scenarios, we setup edge servers with different configurations of available capacity, storage bandwidth, computational power, and network bandwidth between data generators. For storage bandwidth, each edge server can be installed with different number of storage devices (e.g., HDD and SSD) and storage servers, resulting in various storage bandwidths. We are measuring computation power in GB/s, which shows that an edge server ES_j is capable of analyzing certain GB of data in one second. Data centers are connected to data generators with different network connections varying from 1Tb/s to 1Gb/s. Table 6 presents the various values of each parameter for configuration of edge servers.

A. JRAP ANALYSIS IN SIMULATION

Table 7 presents the experimental results for *EE* job request. In this experiment, we use three data generators each initiated a data placement and analysis request; DG_1 initiated a Job_1 of 1TB, DG_2 requested a Job_2 of 800GB whereas DG_3 requested a Job_3 of 500GB after certain time period. For the ease of presentation, in this experiment we are considering $\beta = 1$ (i.e., single edge server is processing complete job). Experimental results in Table 7 reveal that JET is minimum for each job while taking into account all the three resources, i.e., SB, CP, and NB. When Job_1 is executed all the resources are available at each edge server and ES_{14} is the optimal edge server which yields minimum JET. The edge servers that are powerful in one or two resources may not give minimum JET. Table 7 shows the ES_6 has maximum SB of 50Gb/s but still it yields more JET than ES_{14}, because of the slow network connection between data generator and ES_6 resulting in high transfer time t_{tr}. Similarly, ES_{24} and ES_{40} are most powerful in terms of CP and NB, respectively. However, they both have high JET because of the other weak resources. Now, for Job_2 we observe that ES_{14} is not the optimal server because its resources are already being utilized by Job_1. Therefore, JRAP returns ES_{33} optimal for Job_2. Note that, for Job_3 our model selects ES_{28} because it gives minimum JET. It is evident from the experimental results that each parameter can affect the decision making of JRAP. Besides optimal decision, our model also manages the load balancing across edge servers by monitoring resource utilization.

We now discuss performance evaluation for *PO* job category in our simulation $Cluster_{50}$ environment. For this experiment we considered a request of 2TB from a data generator and β is set to 8 by the client. We compare JRAP with two different data placement strategies. The first strategy is the nearest neighbor (NN), which places all data to the closest edge server from the data generator location. The second method is equal distribution (ED), which randomly selects β edge servers from the cluster and equally distribute data in β edge servers irrespective of storage bandwidth. Figure 7(a) shows the results of our experiment: NN shows the worse performance as all the data is stored only in one edge server, whereas ED equally distributes data, resulting in high data placement time for edge servers with less storage bandwidth. JRAP intelligently selects optimal β edge servers and distributes data based on the storage power of each edge server while ensuring that each edge server store the allocated data in almost same time. In our experiment, the edge server with fastest storage bandwidth stores 465GB of data whereas slowest get the share of just 94GB of data from 2TB. These results clearly depict that JRAP outperforms NN and ED methods.

Next, we show the performance evaluation of JRAP for an *AO* job request. In this experiment, we consider the analysis request of 1TB of data, which is distributed among five edge servers ($ES_{14} = 300GB, ES_{34} = 150GB, ES_{19} = 200GB, ES_{30} = 100GB, ES_{43} = 250GB$). We compare performance of JRAP with same two approaches presented above that is NN and ED. In this experiment, the NN approach aggregates all the data to single ES. The ES is selected based on the maximum amount of data stored. Figure 7(b) demonstrate the performance of NN, ED and JRAP approach for *AO* job type. The experimental results reveal that NN shows the worse performance because all the data first aggregated on ES_{14} (because ES_{14} stored maximum data 300GB) and then it performs analysis of entire 1TB of data. ED, equally distributes data without considering the resources on

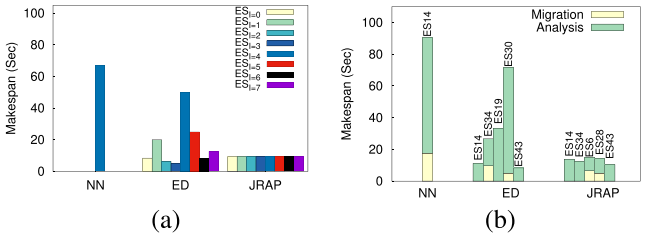

FIGURE 7. Makespan comparison of JRAP with different job distribution techniques on Simulation Testbed. (a) PO job request. ES_i indicates index of edge server selected among β edge servers. (b) AO jobs evaluation.

TABLE 8. Description of real testbed federation configurations.

Parameters	Range
Available Capacity (GB)	600, 800, 1000
Storage Bandwidth (MB/s)	116, 182, 300
Computational Power (MB/s)	35, 75, 115
Network Bandwidth (Gb/s)	IB (56 Gb/s)

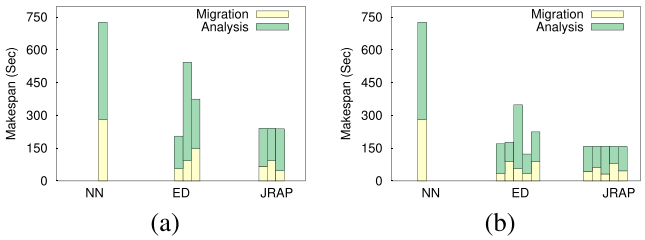

FIGURE 8. Makespan comparison of JRAP with different job distribution techniques for End-to-End job on Real Testbed. (a) with β edge servers = 3. (b) with β edge servers = 5.

edge servers, resulting in unnecessary migration cost and high analysis time. However, JRAP only migrates the data from edge servers with low computational power to minimize the analysis cost. Thus, to reduce the analysis time data from ES_{19} and ES_{30} is migrated to ES_6 and ES_{28}, respectively.

B. JRAP ANALYSIS IN REALISTIC TESTBED

This experiment was performed on a real testbed comprising 10 edge servers. The edge servers were setup with different configurations of resources. Table 8 presents various values of each parameter used to configure edge servers. To evaluate JRAP with real datasets, we downloaded weather forecast datasets from ECMWF [43]. The ECMWF provides a variety of public datasets including operational, reanalysis, and atmospheric composition [43]. We used 50GB of the operational dataset with an end-to-end job workflow, i.e., once dataset storage is completed, GAG application is triggered immediately on the dataset.

Figure 8 shows the performance evaluation of three data placement algorithms; nearest neighbor (NN), equal distribution (ED) and JRAP. It is evident from the results that NN performs the worst because only single edge server is responsible for placing and analyzing the complete job. ED performs better than NN as it utilizes the parallel processing by equally distributing the job workload to β number of

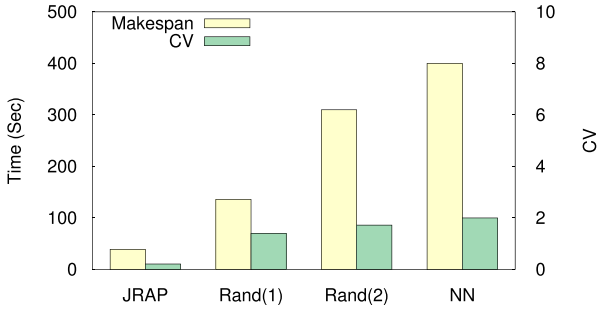

FIGURE 9. Performance comparison of various load balancing techniques with respect to variability of loads across geo-distributed edge servers. We used *Testbed-II* in this experiment.

edge servers. However, it chooses the edge servers randomly and does not consider the resource availability of each edge server. JRAP outperforms both the algorithms because it selects the best β edge servers and also it efficiently distributes the workload based on the storage and computational power of the edge server. For $\beta = 5$, the 12.8GB of data was processed by the fastest server and 8.1GB of data was processed by the slowest server. Also, notice that overall JET for NN is not affected by β because single nearest edge server is selected to execute the complete job. However, the overall JET for ED and JRAP is reduced with the increase in β.

C. IMPACT OF LOAD BALANCING IN JRAP

Figure 9 shows the efficacy of the JRAP for balancing loads across edge servers. For evaluation, we make a job workflow which runs WC, AG, GR and GAG in order. They are initially enqueued in the job queue, and each job is dispatched by a job dispatching algorithm. In order to show the superiority of the JRAP, we compared our approach with different load balancing techniques:

- Random distribution (Rand(1)) of workflow (ES_1 is occupied with GAG application, ES_4 is executing WC, GR and AG).
- Random distribution (Rand(2)) of workflow (ES_2 is responsible for running WC, GAG and AG, ES_4 is running GR).
- Nearest Neighbor (NN) (all workflows are running on ES_2). Experimental results depict that JRAP is responsible for uniformly distributing the workload among the edge servers.

It can be observed from Figure 9 that JRAP performed surprisingly better in load balancing across cluster. Our results proved that the makespan of JRAP is 200 times less than the nearest neighbor approach. Following the same, CV[1] of JRAP is negligible if considered with other load balancing strategies. This distribution can be explained by CV.

[1]The coefficient of variation CV is defined as the ratio of the standard deviation s to the mean m of job loads across edge servers $CV = s$.

VIII. CONCLUSION AND FUTURE WORK

Massive expansion in data generation is leading towards high geo-distributed storage and computation demands. Such geo-located edge servers require federation on top of it to improve data sharing, collaboration and analytics. We investigated and showed the important challenges of federation file systems in particular, global namespace, optimal placement and analysis, data migration and metadata bottleneck. We prototyped the iStore federation file system to emulate and support the feasibility of proposed ideas. We evaluated iStore to show the efficacy of each component via real testbed and simulation framework. The proposed JRAP, DCM and metadata batch-scheme improved the overall federation performance.

ACKNOWLEDGMENT

(Awais Khan and Muhammad Attique are co-first authors.)

REFERENCES

[1] *IDC*. Accessed: Dec. 23, 2018. [Online]. Available: https://www.ibm.com/blogs/internet-of-things/ai-future-iot/

[2] A. Khan, A. Muhammad, Y. Kim, S. Park, and B. Tak, "Edgestore: A single namespace and resource-aware federation file system for edge servers," in *Proc. IEEE Int. Conf. Edge Comput.*, Jul. 2018, pp. 101–108.

[3] L. Krčál and S.-S. Ho, "A scidb-based framework for efficient satellite data storage and query based on dynamic atmospheric event trajectory," in *Proc. 4th Int. ACM SIGSPATIAL Workshop Anal. Big Geospatial Data*, New York, NY, USA, Nov. 2015, pp. 7–14. doi: 10.1145/2835185.2835190.

[4] C. L. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Inf. Sci.*, vol. 275, pp. 314–347, Aug. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025514000346

[5] T. J. Skluzacek, K. Chard, and I. Foster, "Klimatic: A virtual data lake for harvesting and distribution of geospatial data," in *Proc. 1st Joint Int. Workshop Parallel Data Storage data Intensive Scalable Comput. Syst.*, Nov. 2016, pp. 31–36. doi: 10.1109/PDSW-DISCS.2016.010.

[6] Y. Kim, S. Atchley, and G. R. Vallée, and G. M. Shipman, "LADS: Optimizing data transfers using layout-aware data scheduling," in *Proc. 13th USENIX Conf. File Storage Technol.* , 2015, pp. 67–80.

[7] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid datafarm architecture for petascale data intensive computing," in *Proc. 2nd IEEE/ACM Int. Symp. Cluster Comput. Grid*, May 2002, p. 102.

[8] F. Hupfeld *et al.*, "The XtreemFS architecture—A case for object-based file systems in grids," *Concurrency Comput., Pract. Exper.*, vol. 20, no. 17, pp. 2049–2060, Dec. 2008. [Online]. Available: http://dblp.uni-trier.de/db/journals/concurrency/concurrency20.html#HupfeldCKSFHMMC08

[9] A. Rajasekar and R. Moore, *IRODS Primer: Integrated Rule-Oriented Data System* (Synthesis Lectures on Information Concepts, Retrieval, and Services), vol. 2. Morgan and Claypool Publishers, 2010. doi: 10.2200/S00233ED1V01Y200912ICR012.

[10] FedFS. *FedFS-Fedoraproject*. Accessed: May 1, 2017. [Online]. Available: https://fedoraproject.org/wiki/Features/FedFS

[11] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Newton, MA, USA, 2009.

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. 26th Symp. Mass Storage Syst. Technol.*, Washington, DC, USA, May 2010, pp. 1–10. doi: 10.1109/MSST.2010.5496972.

[13] ESnet. *Energy Sciences Network (ESnet)*. Accessed: Dec. 23, 2018. [Online]. Available: http://www.es.net/

[14] 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. 7th Symp. Oper. Syst. Design Implement.*, Nov. 2006, pp. 307–320.

[15] A. Davies and A. Orsaria, "Scale out with GlusterFS," *Linux J.*, vol. 2013, no. 235, Nov. 2013. Art. no. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2555789.2555790

[16] (2017). *Lustre: A Scalable, High-Performance File System*. [Online]. Available: http://cse710.blogspot.kr/2013/02/lustre-scalable-high-performance-file.html

[17] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proc. 24th ACM Symp. Oper. Syst. Princ.*, Nov. 2013, pp. 292–308.

[18] G. B. Brand and A. Lebre, "Gbfs: Efficient data-sharing on hybrid platforms: Towards adding wan-wide elasticity to dfses," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput. Workshop*, Oct. 2014, pp. 126–131.

[19] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, "iFogStor: An IoT data placement strategy for fog infrastructure," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, May 2017, pp. 97–104.

[20] B. Cho and I. Gupta, "Budget-constrained bulk data transfer via Internet and shipping networks," in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, New York, NY, USA, Jun. 2011, pp. 71–80. [Online]. Available: http://dl.acm.org/citation.cfm?id=1998582.1998595

[21] W. Hu, W. Sun, Y. Jin, W. Guo, and S. Xiao, "An efficient transportation architecture for big data movement," in *Proc. 9th Int. Conf. Inf., Commun. Signal Process.*, Dec. 2013, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6782927

[22] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2710–2721, Dec. 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6678116

[23] Z. Er-Dun, Q. Yong-Qiang, X. Xing-Xing, and C. Yi, "A data placement strategy based on genetic algorithm for scientific workflows," in *Proc. 8th Int. Conf. Comput. Intell. Secur.*, Nov. 2012, pp. 146–149. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6405885

[24] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1200–1214, Oct. 2010. doi: 10.1016/j.future.2010.02.004.

[25] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, Apr. 2010, p. 2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855713

[26] P. Teli, M. V. Thomas, and K. Chandrasekaran, "An efficient approach for cost optimization of the movement of big data," *Open J. Big Data (OJBD)*, vol. 1, no. 1, pp. 4–15, 2015. [Online]. Available: http://www.ronpub.com/publications/OJBD_2015v1i1n02_Teli.pdf

[27] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proc. IEEE 19th Annu. Int. Symp. Modelling, Anal., Simulation Comput. Telecommun. Syst.*, Jul. 2011, pp. 227–236.

[28] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in MapReduce," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2012, pp. 419–426.

[29] W. Tantisiriroj, S. Patil, G. Gibson, S. W. Son, S. J. Lang, and R. B. Ross, "On the duality of data-intensive file system design: Reconciling HDFS and PVFS," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2011, pp. 1–12.

[30] S. S. Vazhkudai, X. Ma, V. W. Freeh, J. W. Strickland, N. Tammineedi, and S. L. Scott, "Freeloader: Scavenging desktop storage resources for scientific data," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2005, p. 56.

[31] Y. Yang *et al.*, "Pado: A data processing engine for harnessing transient resources in datacenters," in *Proc. 20th Eur. Conf. Comput. Syst.*, Apr. 2017, pp. 575–588.

[32] S. V. Travis Bar, N. Langfeldt, and T. McNeal. *Linux Nfshowto*. Accessed: Aug. 25, 2002. [Online]. Available: http://nfs.sourceforge.net/nfshowto/

[33] B. K. R. Vangoor, V. Tarasov, and E. Zadok, "To FUSE or not to FUSE: Performance of user-space file systems," in *Proc. 15th USENIX Conf. File Storage Technol.* Santa Clara, CA, USA, 2017, pp. 59–72. [Online]. Available: https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor

[34] A. Rajgarhia and A. Gehani, "Performance and extension of user space file systems," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, Mar. 2010, pp. 206–213. doi: 10.1145/1774088.1774130.

[35] GoogleDevelopers. *gRPC: Google Remote Procedure Call*. Accessed: Dec. 23, 2018. [Online]. Available: http://www.grpc.io/

[36] *Protocol Buffers Âđ|Âđ Google Developers*. [Online]. Available: https://developers.google.com/protocol-buffers/

[37] SQLite. *SQLite Home Page*. Accessed: Dec. 23, 2018. [Online]. Available: https://www.sqlite.org/

[38] L. Xu, H. Jiang, L. Tian, and Z. Huang, "Propeller: A scalable real-time file-search service in distributed systems," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun./Jul 2014, pp. 378–388.
[39] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, scalable metadata search for large-scale storage systems," in *Proc. 7th Conf. File Storage Technol.*, Feb. 2009, pp. 153–166. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525920
[40] L. Xu, Z. Huang, H. Jiang, L. Tian, and D. Swanson, "VSFS: A versatile searchable file system for HPC analytics," Dept. Comput. Sci. Eng., Univ. Nebraska-Lincoln, Lincoln, NE, USA, Tech. Rep. 128, 2013.
[41] Ext4. *Ext4 Documentation*. Accessed: Dec. 23, 2018. [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/ext4.txt
[42] LLNL. *Llnl/mdtest: Used for Testing the Metadata Performance of a File System*. Accessed: Dec. 23, 2018. [Online]. Available: https://github.com/LLNL/mdtest
[43] *ECMWF*. Accessed: Dec. 31, 2018. [Online]. Available: https://www.ecmwf.int/en/forecasts/datasets

AWAIS KHAN received the B.S. degree in bioinformatics from Mohammad Ali Jinnah University, Islamabad, Pakistan. He is currently pursuing the Ph.D. degree (integrated program) with Sogang University, Seoul, South Korea. He was with one of the leading software companies as a Software Engineer, from 2012 to 2015. He is currently a member of the Laboratory for Advanced System Software, Computer Science and Engineering Department, Sogang University. His research interests include cloud computing, cluster-scale deduplication, and parallel and distributed file systems.

MUHAMMAD ATTIQUE received the bachelor's degree in information and communication systems engineering from the National University of Science and Technology, Pakistan, in 2008, and the Ph.D. degree from Ajou University, South Korea, in 2017. He is currently an Assistant Professor with the Department of Software, Sejong University, South Korea. His research interests include location-based services, spatial queries in the road networks, and big data analysis.

YOUNGJAE KIM received the B.S. degree in computer science from Sogang University, South Korea, in 2001, the M.S. degree from KAIST, in 2003, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, University Park, PA, USA, in 2009. He was a Staff Scientist with the U.S. Department of Energy's Oak Ridge National Laboratory, from 2009 to 2015, and an Assistant Professor with Ajou University, Suwon, South Korea, from 2015 to 2016. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. His research interests include distributed file and storage, parallel I/O, operating systems, emerging storage technologies, and performance evaluation.