EdgeStore: A Single Namespace and Resource-aware Federation File System for Edge Servers

Awais Khan[1], Muhammad Attique[2], Youngjae Kim[1], Sungyong Park[1], Byungchul Tak[3]
[1]Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea
[2]Department of Software, Sejong University, Seoul, Republic of Korea
[3]Department of Computer Science and Engineering, Kyungpook National University, Daegu, Republic of Korea
{awais, youkim, parksy}@sogang.ac.kr, attique@sejong.ac.kr, bctak@knu.ac.kr

***Abstract*—With the increasing adoption of edge computing, the capacity requirements of the edge servers are also growing. Especially the data volume generated from a large number of edge clients and/or edge devices demand more capacity to be able to store them for processing. The growing gap between the data volume and current storage capacity is motivating the need towards building aggregated storage spaces. Aggregated storage can be an effective way to extend edge servers' overall storage capacity by combining storage resources of other nodes under the agreement to share. Several Federation file systems exist to meet this aggregate storage needs but are not without limitations. Dependency to the specific software stack makes it unfit for general-purpose use and they often neglect important features critical for the performance.**

In this paper, we address the important challenges of building the Federation on top of edge servers with the heterogeneous file system and resource configurations. We prototyped EdgeStore, a Federation File System for Edge Servers. EdgeStore equips the users with an aggregate storage namespace and federates resources of edge servers, to enable high resource-sharing in Federation. We propose, Job and Resource-Aware Request Placement algorithm (JRAP) to take advantage of edge server resource heterogeneity. To evaluate the usefulness of EdgeStore, we consider two federation scenarios i) with same resource configurations and ii) with different resource configurations. We evaluate the efficacy of various big data applications from data storage to analysis using EdgeStore on a real testbed.

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 [7]. 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 [8]. This untamed growth of data volume poses serious challenges to Big Data applications, requiring massive scale analytical systems [9]. 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. The 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. However, the current notion of federation tends to assume homogeneous architectures, i.e., all edges across the network are required to have a uniform specifications such as identical file systems, storage bandwidth, compute power and network connectivity. Existing studies in this space include Xtreem file system (XtreemFS) [3], Grid Data Farm (GFarm) [10], and HDFS [5]. However, these file systems allow the aggregation of only the identical file system. For example, GFarm file system can aggregate another edge servers that are formatted with the same GFarm file system [10]. Such narrow requirements result in underutilizing the CPU, memory and storage resources.

Another challenge in building the federation of heterogeneous edge servers is designing optimal job placement algorithm that ensures fair load-distribution across participating edge servers. We generalize this problem as a Request Placement Problem in Federation environments. A recent study, IFogStor [11], addresses similar problem at the network layer for the data placement of IoT devices. However, the problem becomes more challenging in the federation because there exist different type of Big Data applications, and the resource heterogeneity of the edge servers. Big Data applications can be broadly categorized as storage-intensive or compute-intensive. Storage-intensive applications require large storage bandwidth (e.g., storing weather datasets) whereas, compute-intensive requires high CPU power (e.g., running analysis on the stored weather datasets). Suboptimal placement of requests to the edge servers may not only incur bad performance, but also huge data movement costs wasting scarce network bandwidth. For example, a user or an application requires to store and run analysis on data. The edge server selected for data storage has very low computational power and can increase the analysis application latency to high extent. In such cases, additional data movement cost is incurred.

978-1-5386-7238-9/18/$31.00 ©2018 IEEE
DOI 10.1109/EDGE.2018.00021

Federation File System	***Placement***	***POSIX***	***Federation Type***	***Migration***	***Metadata***
GBFS [1]	Random	Posix	P/DFS	No Awareness	Central
GFarmFS [2]	Random	Posix	GFarm	No Awareness	Central
XtreemFS [3]	Random	Posix	XtreemFS	No Awareness	Central
iRODS [4]	Rule-based	Posix	iRODS	No Awareness	Central
Hadoop [5]	Random	Non-Posix	HDFS	Migration-Aware	Central
FedFS [6]	Random	Posix	FS Referrals	No Awareness	Central
EDGESTORE	JRAP	Posix	All Posix	Migration-Aware	Central

TABLE I: List of different features of Federation File Systems. EDGESTORE implements central metadata however, batch-based metadata scheme is adapted to reduce metadata contention.

To address these challenges of building the Federation of heterogeneous edge servers, we propose an architecture of single aggregate namespace with the capability to integrate edge servers formatted with different POSIX-compliant file systems. The aggregate namespace unifies the storage resources of networked edge servers so that Big Data applications can directly execute within a single file namespace. To cater for the effective resource management in the Federation, we designed Job and Resource-aware Request Placement algorithm (JRAP). The algorithm takes into account the job type, and available resource configurations such as storage, compute and network bandwidth at each edge server. The objective is to compute Job Execution Time (JET) of edge servers and to route the requests to edge servers with minimum expected job execution time. Since algorithm requires information of all the edge servers, we need to manage various metadata of the Federation. Metadata we handle are: the total number of edge servers, resource configuration, which files are stored where and job statistics at each edge server. We have designed a lightweight batch-based metadata manager to reduce metadata contention in I/O intensive applications.

In this work we make the following contributions:

- We investigate and highlight key challenges required in building the Federation atop of edge servers. To the best of our knowledge, these challenges are not considered collectively in previous studies.
- We prototype EDGESTORE, a single namespace file system capable of aggregating the resources of edge servers. Single namespace allows a unified view of all the resources and to centrally manage resources using batch-based metadata manager.
- To ensure load-distribution and effective utilization of edge server resources, we design the Job and Resource-aware Request Placement algorithm ($JRAP$). JRAP computes the optimal data center providing minimal job execution time (JET) based on the job types. We provide a comprehensive modeling of request placement problem. JRAP algorithm is general enough that it can be applied to any existing Federation file systems.
- To evaluate EDGESTORE, 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 EDGESTORE on the real testbed.

II. RELATED WORK

Cloud providers offer diverse set of commercial cloud storage services to meet the needs of cloud users. Some of the well-known services include Skydrive, Dropbox, iCloud and Google drive. Most of them operate as single stand-alone offerings with differing goals and requirements. On the other hand, researchers have looked at building the federation of cloud storages distributed over multiple geo-distributed data centers equipped with high-speed networks. However, these federated file systems are designed with specific requirements, application types and/or software architectures. Table I lists key features of several Federation file systems.

The parallel and distributed file systems such as Ceph [12], Gluster [13] and Lustre [14] are designed for a single-site installations, i.e. only for a single cluster. Grid Data Farm (GFarm) [10], Xtreem File System (XtreemFS) [3] and HDFS [5] are specifically designed for petascale storage and computing. SPANStore [15] introduces a geo-replicated storage service that focuses on the cost-effectivenss and delivering the key-value store service. GBFS [1] 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 [6] and iRods [4], 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.

Major Federation file systems in Table I employ random request placement except for iRods [4]. However, random placement is always not an efficient solution. It incurs 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. IFogStor [11] implements a resource-aware placement methodology for IoT data placement. Agarwal et al. [16] proposed an automated data placement mechanism, *Volley*, for geo-distributed cloud services. Yuan et al [17] proposed a data placement-based approach for scientific workflows. Apart from these, there are very few studies conducted on scavenging existing available resources such as desktop and server machines. FreeLoader [18] and Pado [19] introduce the

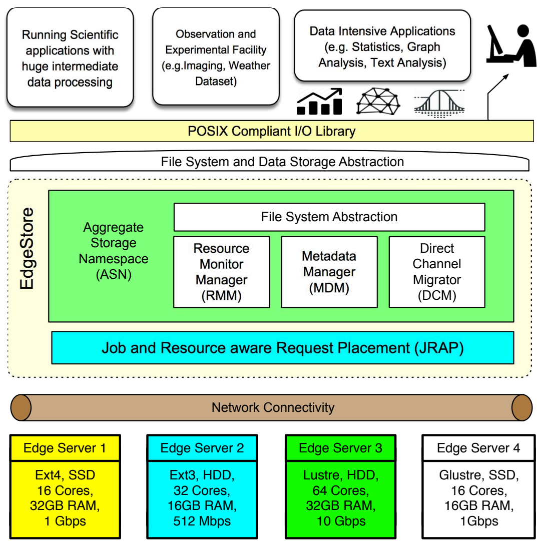

Running Scientific applications with huge intermediate data processing
Observation and Experimental Facility (e.g.Imaging, Weather Dataset)
Data Intensive Applications (e.g. Statistics, Graph Analysis, Text Analysis)
POSIX Compliant I/O Library
File System and Data Storage Abstraction
EdgeStore
Aggregate Storage Namespace (ASN)
File System Abstraction
Resource Monitor Manager (RMM)
Metadata Manager (MDM)
Direct Channel Migrator (DCM)
Job and Resource aware Request Placement (JRAP)
Network Connectivity
Edge Server 1
Ext4, SSD 16 Cores, 32GB RAM, 1 Gbps
Edge Server 2
Ext3, HDD, 32 Cores, 16GB RAM, 512 Mbps
Edge Server 3
Lustre, HDD, 64 Cores, 32GB RAM, 10 Gbps
Edge Server 4
Glustre, SSD, 16 Cores, 16GB RAM, 1Gbps

Fig. 1: EDGESTORE Architecture. Edge servers show heterogeneity in file system and resource configurations.

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.

There are several key differences in our study as compared to existing Federation file systems. The objective of EDGESTORE is to enable Federation on top of heterogeneous nodes such as desktops or server machines acting as network edges in the data center topology. The file system heterogeneity is never considered in prior studies and none of the existing file systems allow aggregation of disparate type of file system. Moreover, all of the existing studies target Cloud and not the Federation environments where resources fluctuate highly. Also, none of the existing studies consider the application, job, and resource-awareness all together in the request placement model. We envision EDGESTORE to provide aggregate storage namespace atop of edge servers connected via network.

III. EDGESTORE: SYSTEM OVERVIEW

In this section, we present overall architecture of EDGESTORE followed by description of key components.

A. EDGESTORE *Architecture*

Figure 1 shows the architectural overview of EDGESTORE, aggregating edge servers connected via the network and equipped with different file systems.

The Aggregate Storage Namespace (ASN) in EDGESTORE provides location-transparency and is responsible for equipping EDGESTORE with POSIX standards and file system operations. The file systems of edge servers are mounted on a user machine via linux NFS [20]. The ASN layer of each client builds a virtual abstraction which keeps the users unaware of the actual data location and underlying storage architecture. As such, unification of multiple edge servers requires some additional metadata management such as identities of participating edge servers, resource details and file-to-edge server mappings, i.e., what file is stored at which edge server. To accommodate such metadata, we designed metadata manager (MDM) which records all the file-to-edge server mappings and Federation metadata such as total edge servers in the Federation, resource statistics, on-going jobs and operations in the ASN layer. To collect the resource availability at each edge server, we deploy the Resource Monitor Manager (RMM). RMM can periodically collect information of resource availability at each edge server.

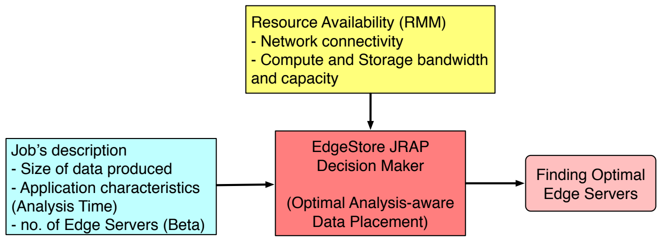

Resource Availability (RMM)
- Network connectivity
- Compute and Storage bandwidth and capacity
Job's description
- Size of data produced
- Application characteristics (Analysis Time)
- no. of Edge Servers (Beta)
EdgeStore JRAP Decision Maker
(Optimal Analysis-aware Data Placement)
Finding Optimal Edge Servers

Fig. 2: Workflow illustration of user interaction and optimal edge server selection in EDGESTORE.

Then, based on the resource availability condition, Job and Resource aware Request Placement (JRAP) algorithm determines the optimal target edge server. We also deploy the Direct Channel Migrator (DCM), a migration protocol which allows data migration from one edge server to another edge server bypassing the Federation namespace for efficiency.

B. *Job Execution Workflow*

Job and Resource-aware Request Placement (JRAP) is the core part of EDGESTORE. When a request arrives at the namespace layer, ASN invokes the JRAP to find a best-fit edge server with minimum job execution time for the request. JRAP algorithm computes the optimal edge server based on multiple parameters, received as input from other components. Figure 2 depicts a general workflow of user interaction and edge server selection in EDGESTORE. The user provides a complete job description, such as job type, total data size, application characteristics, and number of edge servers to complete the job. This information is used to estimate the application runtime. Application time can be predicted through regression with past experimental results stored in MDM such as application type, analysis time, data size, and compute power as experimental history of EDGESTORE. RMM provides the snapshot of available resources at edge servers, i.e., network bandwidth, available storage capacity, storage bandwidth and compute power of each edge server. JRAP takes into account these inputs and determines the optimal edge server. Then, EDGESTORE routes the request to that particular edge server.

In this study, we focus on building the Federation with commodity edge servers equipped with dissimilar file systems.

After the data is generated, the user inquires MDM at the application level, finds the location of the files, and connects to the edge server to launch the analysis application. This remote execution can be done through a job scheduler implementation, or it can be done manually by the user. The current work includes no integration of remote execution layer with EDGESTORE. This study assumes that this remote execution is done either way.

IV. EDGESTORE: JOB AND RESOURCE AWARE PLACEMENT

In this section, we present our job classification, system model, and algorithm for optimal request placement in Federation comprising of heterogeneous edge servers.

A. Problem Formulation: Objective and Constraints

While Big Data is now in vogue, many organizations are using the private cloud-based Federations for their applications. Each application has different job type 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 II.

Job Category	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 II: The job categorizations for Big Data applications.

For the convenience of the readers, major notations used in this paper are listed in Table III. We consider a network connected set of edge servers $ES = \{ES_1, ES_2, ..., ES_n\}$ and a set of data generators (e.g., weather sensing satellites, or clients) $DG = \{DG_1, DG_2, ..., DG_n\}$ that are continuously generating large volumes of data. The data generator connects to the edge server via dedicated high bandwidth network with S VPN switches at the user side and S' VPN switches each co-located with edge server.

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 servers in the Federation may give a minimum job execution time however it increases the data migration and metadata overheads. Therefore, we introduced a parameter β, which is a positive real number ($\beta \in \mathbb{R}^+$) and it controls the number of edge servers used for particular request. Our algorithm finds the set of β number of optimal edge servers. By default, we consider β as a single edge server unless specified by the user. The workload is proportionally distributed among edge servers based on the resources availability of edge server. Specifically, the edge server with powerful resources gets the maximum proportion. 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 a single edge server.

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

Constants	Description
$G(N, E, W)$	The graph model of a Federation infrastructure.
W(e)	Weight of a link connecting two points.
ES	The set of edge servers.
DG	The set of data generators or users.
α_k	Size of slice k.
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.

TABLE III: Summary of notations used in model formulation.

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. Therefore, the total amount of data $x_{ij} = p.\alpha_k$. The time taken to transfer one slice of data from DG_i to ES_j is denoted as n_{ij}. Therefore, the total data transfer time is given as:

$t_{tr} = \sum_{DG_i \in DG, ES_j \in ES} p.n_{ij}$

Storage Time: It is an important factor to be considered in choosing the edge server for request placement. Let's consider s_j be the time taken to store one slice of data by edge server ES_j. The overall storage time incurred is:

$t_{st} = \sum_{ES_j \in ES} p.s_j$

Analysis Time: The edge 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 overall analysis time is represented as:

$t_{an} = \sum_{ES_j \in ES} p.r_j$

B. Job and Resource-aware Request Placement

In this section, we present our placement model with respect to three job categories as shown in Table II.

End-to-End Job (EE): This job type requires both data placement and immediate analysis. 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 straight forward way to reduce JET is by deploying the high-speed network between the data generator and edge servers. However, simple network improvements only increase the data transfer speed, and not end-to-end data placement and analysis time. Therefore, the storage bandwidth and computational power of edge server also plays a vital role in achieving the minimum JET. In addition, JRAP also needs to consider the current workload and the availability of resources at the edge servers. 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 and computational power.

Assume the data generator $DG_i \in DG$ has initiated a 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 DC_j.

$$JET_{ES_j} = max(t_{tr}, t_{st}) + t_{an} \tag{1}$$

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

Our objective function is to determine a set of optimal edge server that minimizes the overall job execution time. Therefore, the job request is sent to β edge servers simultaneously and the maximum JET consumed by any edge server is taken as overall JET. We can represent it as:

$$JET = max(JET_{ES_1}, JET_{ES_2}, ..., JET_{ES_\beta}) \tag{2}$$

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 server edge simultaneously. Therefore, the maximum data placement time consumed by any edge server considered as aggregated T_{pl}. We can represent it as:

$$T_{pl} = max(t_{pl_1}, t_{pl_2}, ..., t_{pl_\beta}) \tag{3}$$

Analysis Only (AO): For data analysis only jobs, we adopt an on-site analysis approach, whose basic idea is to perform analysis on the edge server unless migrating data to any other edge server improves analysis time. Therefore, the data migration occurs only in two cases: (1) Edge server 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.

The total analysis time T_{an} of ES_j can be formulated as: $T_{an_j} = t_{mi} + t_{an}$

In 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 server:

$$T_{agg_{an}} = max(T_{an_1}, T_{an_2}, ..., T_{an_n}) \tag{4}$$

C. Job and Resource-aware Request Placement Algorithm

In this section, we present Job and Resource-aware Request Placement algorithm (Algorithm 1), that can manage each job based on their type. To simplify the presentation, we consider EE job category in the following algorithm description. 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

Algorithm 1: Optimal Job and Request Placement Algorithm.

```
Input: $DG$, list of data generator; $ES$, list of edge servers; $x_{ij}$,
    requested data size; $\beta$, number of edge servers used for data
    placement; JT, Job type
Output: $ES_{optset}$, /*Optimal edge server set*/
1  $ES_{sort} \leftarrow sort.ES(JET_{ES_j}, JT)$ /*sort ES list by Job Execution
     Time*/
2  $ES_{candopt} \leftarrow ES_{sort_i=1to\beta}$ /*optimal set contains first $\beta$ edge
     servers from sorted list*/
3  $allocate.size(ES_{candopt})$
4  $notifyresource_{usage}(ES_{candopt})$
5  $ES_{candopt}.JET$ = $ComputeJET(ES_{candopt,JT})$
6  $findopt = false$
7  while $findopt \neq true$ do
8      for $each\ ES_j\ in\ ES_{candopt}$ do
9          if $ES.availabcap > allocate.size$ then
10             $JET_{opt} \leftarrow JET$
11             $ES_{optset} \leftarrow ES_{candopt}$
12             $findopt = true$
13         end
14         $JET$ = $ComputeJET(ES_{candopt_{size}})$
15         $ES_{victim} \leftarrow ES_{candopt}.pop()$ /*pop ES which does not
             have required available capacity*/
16         $ES_{candopt} \leftarrow ES_{sort}.push()$ /*push next ES in
             $ES_{candopt}$ from sorted list*/
17         $JET_{new}$ = $ComputeJET(ES_{candopt})$
18         if $JET_{new} < JET$ then
19             $JET_{opt} \leftarrow JET_{new}$
20             $ES_{optset} \leftarrow ES_{candopt}$
21             $findopt = true$
22         end
23     end
24 end
25 return $ES_{optset}$
```

decision making of algorithm depends on the current resource availability at each edge server. At first, algorithm sorts the edge server based on the JET_{ES_j} of each server and generates the candidate optimal set $ES_{candopt}$ by taking first β severs from the sorted list. We then distribute the data among network connected edge servers based on the available resources and computes 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 case, the algorithm assigns the remaining data to other edge server 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, algorithm terminates by returning ES_{optset} with minimum JET. To achieve minimum JET considering parallel storage and analysis, we slightly modified the Algorithm 1 to find the set of optimal edge servers. 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 (3) 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 (4) to compute the $T_{agg_{an}}$.

V. EdgeStore: Design and Implementation

In this section, we describe our rationale behind the ASN design and implementation, Job and resource-aware request placement (JRAP) and batch-based metadata manager (MDM).

A. ASN: Aggregate Storage Namespace

We prototyped ASN in FUSE's high-level API v2.9.4 [21]. We implemented all the basic file system operations in ASN such as *init, access, create, getattr, mkdir, read, readdir, write* When EDGESTORE is mounted, *asn_init* is the first function to execute. The metadata manager address is supplied as an argument at mount time. *asn_init* performs initial setup configuration checks, and validate the metadata manager (MDM) service is up and running. Then, it initiates a request to MDM by providing a list of participating edge servers in Federation with their configurations pre-defined in EDGESTORE. Upon receiving a response from MDM, *asn_init* fills the private data structures to keep the list of edge servers contributing resources in EDGESTORE Federation. On the contrary, *asn_destroy* unmounts EDGESTORE. The metadata-oriented methods like *ufs_create* and *asn_open*, all require MDM assistance in order to get the data location in Federation, i.e., which edge server contains the data. When data-oriented methods such as *asn_write* are invoked, only a 4KB of data is sent to the userspace daemon for writing but when *big_writes* optimization parameter is passed as an argument to EDGESTORE, then bigger chunks of data are sent to the userspace daemon. This chunk size value is configured in FUSE configurations. We used *max_writes* of size 128KB in order to observe better and consistent performance throughout our implementation and evaluation.

B. MDM: Metadata Manager

Metadata manager (MDM) in our design is of vital importance. Firstly, it interacts with all the EDGESTORE components. Secondly, all requests require assistance from the MDM to complete the operations. Thirdly, the federation metadata statistics such as the total number of edge servers, with default resource configurations are managed by the MDM. Such significance motivated us to design a lightweight and efficient batch-based metadata approach, where the number of metadata I/Os can be reduced and which in turns improves the EDGESTORE performance.

We implemented MDM using gRPC, a high performance, open-source, multi-platform, language neutral RPC framework for developing distributed applications and services [22]. MDM is defined as a gRPC service. gRPC uses google protocol buffers (Protobuf) for underlying message interchange format [22]. Protobuf is google's open source mechanism for serializing structured data [23]. We defined standard messaging format in Protobuf for communication among all the components. The file system metadata e.g., stat, size, date are maintained by the edge server file system. Our batch-based scheme holds the metadata in EDGESTORE memory and do not communicate with the MDM unless triggered. Once triggered, we generate a single multi-valued insert query and send the message to MDM. In this way, we reduce the multiple I/Os to MDM. We use two types of triggers to flush the cached metadata to MDM. First trigger is, user controlled batch size and second is time-based trigger. If batch size is not defined, certain number of requests stay in cache unless timer expires and flushes the metadata using same single SQL insert query. This batch-scheme is designed to cache insert and update metadata requests only. However, there is a fault-tolerance issue related with this batch-based scheme but we believe that, it can be omitted by using persistent memory storage such as Flash or PRAM. We used SQLite [24] to store metadata.

C. JRAP: Job and Resource-aware Request Placement

JRAP is responsible for controlling and load-balancing the request placement in the Federation built on top of resource heterogeneous edge servers. JRAP depends on a few parameters to calculate the optimal edge server at request arrival. The parameters include job type, available capacity, storage bandwidth, computation power, and network bandwidth. The available resource values are provided by the RMM running on each edge data center. Once all resource values are input to JRAP, JRAP filters the edge server list based on job type and available capacity. Then JRAP computes the JET on filtered edge server list. JRAP output is optimal data center with minimum job execution time. The output is sent to MDM to store the file and its location. The JRAP is implemented as part of ASN, where all data related-operations *create*, *write* and *read* consult JRAP to provide optimal edge server, which can complete job in minimum time. Currently, JRAP uses the snapshot decision model based on the information available at the start of the job. JRAP modeling and algorithm details are listed in Section IV.

D. DCM: Direct Channel Migrator

Direct channel migrator (DCM) is accountable for data movement across the edge servers in the Federation. The major reason to implement a direct channel communication is to optimize the task completion time. If we fetch the data first through Federation namespace and then, copy the data to destination edge server. It requires copying round-trip that can be omitted via direct channel migration approach. We implemented the DCM by extending an end-to-end data transfer software [25]. The DCM is integrated with JRAP. Whenever, JRAP requires data migration, it triggers DCM service on the source with a complete transfer request format. A simple request includes data location (edge server hosting data), data size, destination edge server (edge server to which data will be transferred).

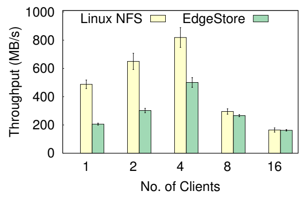
VI. EVALUATION

A. Evaluation Setup

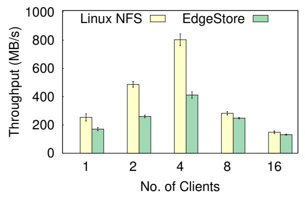
1) Testbeds: We evaluated EDGESTORE on two different testbeds shown in Table IV. The Testbed I is homogenous and comprises of 4 edge servers connected via Infiniband (56Gbps), whereas Testbed II is heterogeneous. We use 4 desktop machines with varying resource configurations at each edge server as shown in Table IV (Testbed II). The edge servers are mounted on the data generator using Linux

Testbed	**Nodes**	**Disk** (MB/s)	**CPU** (GHz)	**Network** (Mb/s)
Testbed I	4 ESs	81	2.60 x16 Cores	IB(56 Gb/s)
Testbed II	ES 1	116	2.40 x8 Cores	942
	ES 2	86	3.20 x8 Cores	239
	ES 3	182	2.2 x8 Cores	91
	ES 4	116	1.7 x4 Cores	942

TABLE IV: Description of testbed setup used for evaluation.



(a) Big files



(b) Small files

Fig. 3: Performance analysis of EDGESTORE on Testbed I.

NFS [20]. We build Ext4 Federation via EDGESTORE prototype. There are two reasons to use the Ext4 file system Federation. First, we have a limited small-scale testbed and second, it is widely used and has a well-documented design which facilitates performance analysis. Before every experiment, we drop system cache and remount the EDGESTORE. EDGESTORE is implemented in C++ comprising of 3000 lines of codes. To fairly evaluate our prototype, we used four realistic Big Data applications namely *Group-by Aggregation* (GAG), *Aggregation* (AG), *Grep* (GR) and *Word-Count* (WC). The analysis time of above mentioned applications with different data sizes were pre-collected and stored in MDM.

B. EDGESTORE *Performance Analysis*

To analyze performance overhead, we compare EDGESTORE with Linux NFS. This experiment is conducted on 2 nodes of Testbed I. We mount EDGESTORE on top of NFS client. We use two types of workloads i) Big Files (16 files x 1GB) and ii) Small Files (1600 files x 10MB) to observe FUSE layer and metadata overhead. Figure 3(a) and (b) shows the throughput for big and small files workload. We increase the number of clients in each experiment. The peak performance observed by NFS is 800MB/s whereas, EDGESTORE is 450MB/s with 4 clients. However, when we use 16 clients, NFS shows huge performance degradation (70%) as compared to NFS with 4 clients. We believe this as network congestion by Linux NFS. Whereas, EDGESTORE shows degraded performance mainly due to FUSE implementation overhead. It is because FUSE uses buffer space and OS page-cache. To verify our results, we also conducted same experiments with IOR [26] and observed the same performance patterns. The error bars in Figure 3(a) and (b) depicts the ratio of standard deviation to the mean for multiple runs of the experiment.

C. Evaluating the Efficacy of JRAP in EDGESTORE

Figure 4(a) presents the comparison of JRAP with different placement approaches used in file systems. This experiment is conducted on Testbed I. To analyze the job distribution in EDGESTORE, we created three job sets, one set per each job category EE, PO and AO as defined in Section IV. The first job set comprises of four EE jobs, the second includes four PO jobs and third, contains four AO jobs. We use 10GB of job workload. Each job set is evaluated via Round-Robin (RR), Random (Rand) and JRAP placement approach in EDGESTORE. Figure 4(a) shows the total completion time of all jobs.

We observed that JRAP outperformed the RR and Rand. The RR assigns the first job of each job set to ES_1, ES_2 and ES_3 without considering job type and resources at the edge server. In such case, the ES_2 in Federation is under-utilized whereas, ES_1 is over-utilized hence, the response time is considered as the maximum time taken by the edge server. Similarly, Rand distributes the job sets randomly across the edge servers. The results showed that maximum time taken by Rand is less than RR but, higher than JRAP. The JRAP approach shows the balanced utilization of resources by assigning jobs to appropriate edge servers as per job type and resource availability. The evaluation showed the significance of request placement as a vital feature in heterogeneous Federation file systems.

Figure 4(b) presents the JET of all four workloads of size 1GB with and without the integration of JRAP in EDGESTORE. The reason to use smaller datasets is to clearly show the overhead and effectiveness of EDGESTORE. To show the efficient decision making and resource management of JRAP, we consider all four jobs are simultaneously generated by a data generator. The NoJRAP refers to EDGESTORE without JRAP. NoJRAP schedules request placement in a Round-Robin fashion i.e., WC, AG, GR and GAG are assigned to ES_1, ES_2, ES_3 and ES_4, respectively. Then, it aggregates all data to ES_2 (because ES_2 has the highest computational power as shown in Table IV) for data analysis which incurs additional migration overhead. The labels at the top of each bar represent the edge server participating in the job.

Figure 4(b), demonstrates that JRAP always made an optimal decision by efficiently utilizing the available resources at each edge server ensuring minimum JET. We observed that placement time and analysis time for WC-NoJRAP is smaller than WC-JRAP. However, due to migration overhead commulative JET of WC-NoJRAP is higher than WC-JRAP. Similar trend is observed for other applications where NoJRAP can outperform JRAP in individual placement or analysis time. However, JRAP shows superiority in terms of end-to-end analysis aware job execution time. We observe from experimental results that storage, computation and network bandwidth all together play important role in efficient end-to-end request placement The decision making of JRAP exploits this observation to deliver the optimal decision with minimum JET.

Figure 4(c) shows the analysis of (AO) job evaluation with direct channel migration in Federation with a certain workflow. This experiment is conducted on Testbed II. We defined a workflow in this experiment, the edge servers ES_1 and ES_2 can only run analytical jobs in Federation because of high

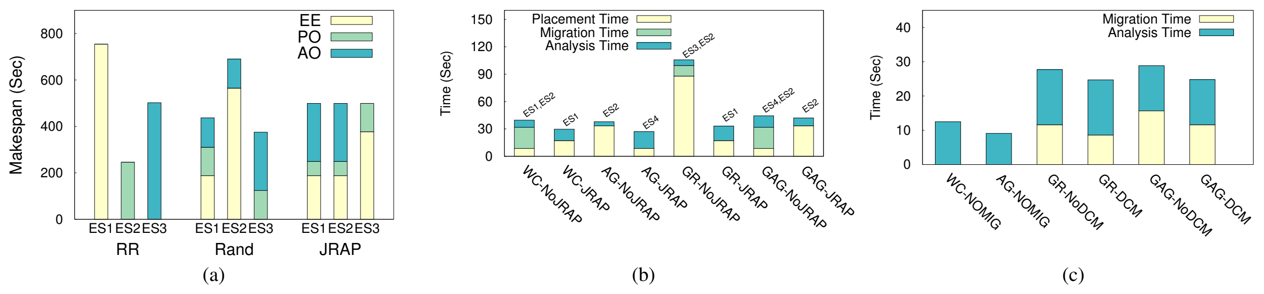


Fig. 4: Efficacy of JRAP in EDGESTORE with β=1. (a) JRAP comparison with other job placement approaches evaluated in Testbed-I, (b) refers to evaluation of EE jobs on Testbed-II and (c) AO jobs in Federation evaluated on Testbed-II with and without DCM.

computation power. We first stored 1 GB of data on each edge server including ES_1 and ES_2. We conducted this experiment to show the effectiveness of data migration in Federation. In Figure 4(c) WC and AG are executed without any migration because data is already stored on ES_1 and ES_2, whereas, data stored on ES_3 and ES_4 has to be migrated to either analytical edge server. The JRAP manager determines the destination data centers to migrate data to ES_1 and ES_2 based on job type and resources. The NoDCM in this experiement refers to no direct channel migration and requires data to be transferred via Federation. The data first needs to be collected at EDGESTORE Federation and then, transfer to required edge server where, analysis will be performed. The experimental results depict that, even in small scale Federation, data migration can impact the performance.

VII. CONCLUSION

In this work, we have designed and developed EDGESTORE capable of providing unified view of heterogeneous file systems and resources in edge servers. In order to achieve the single aggregate storage namespace, EDGESTORE employes various techniques such as Job and Resource-aware placement algorithm (JRAP), efficient metadata management and Fuse-based file operation support. We have evaluated EDGESTORE on homogeneous and heterogeneous real testbed and simulation environments using different various Big Data applications. Our evaluation supports the feasibility of EDGESTORE Federation on top of commodity edge servers. JRAP algorithm also outperforms the other placement approaches.

ACKNOWLEDGEMENT

This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. B0101-17-0644, Research on High Performance and Scalable Manycore OS) and Institute for Information & communication Technology (IITP) grant funded by the Korea government (MSIT) (No. 2015-0-00590, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development).

REFERENCES

[1] G. B. Brand and A. Lebre, "Gbfs: Efficient data-sharing on hybrid platforms: Towards adding wan-wide elasticity to dfses," in *2014 International Symposium on Computer Architecture and High Performance Computing Workshop*, 2014.

[2] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Computing*, 2010.

[3] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario, "The xtreemfs architecture - a case for object-based file systems in grids."

[4] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, *iRODS Primer: Integrated Rule-Oriented Data System*.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10.

[6] FedFS, "Fedfs-fedoraproject," https://fedoraproject.org/wiki/Features/FedFS.

[7] IDC, https://www.ibm.com/blogs/internet-of-things/ai-future-iot/.

[8] L. Krčál and S.-S. Ho, "A scidb-based framework for efficient satellite data storage and query based on dynamic atmospheric event trajectory," ser. BigSpatial'15, 2015.

[9] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information Sciences*, vol. 275, pp. 314 – 347, 2014.

[10] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid datafarm architecture for petascale data intensive computing," in *Cluster Computing and Grid, 2002.*

[11] M. Naas, P. Raipin Parvedy, J. Boukhobza, and L. Lemarchand, "ifogstor: An iot data placement strategy for fog infrastructure."

[12] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '2006.

[13] A. Davies and A. Orsaria, "Scale out with GlusterFS," *Linux J.*, vol. 2013, no. 235, Nov. 2013.

[14] Lustre: A Scalable, High-Performance File System, http://cse710.blogspot.kr/2013/02/lustre-scalable-high-performance-file.html.

[15] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13.

[16] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-distributed Cloud Services," *7th USENIX Conference on Networked Systems Design and Implementation*, 2010.

[17] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A Data Placement Strategy in Scientific Cloud Workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.02.004

[18] 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 *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, 2005.

[19] Y. Yang, G.-W. Kim, W. W. Song, Y. Lee, A. Chung, Z. Qian, B. Cho, and B.-G. Chun, "Pado: A data processing engine for harnessing transient resources in datacenters," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17.

[20] S. V. Travis Bar, Nicolai Langfeldt and T. McNeal, "Linux NFS HOWTO," http://nfs.sourceforge.net/nfshowto/.

[21] B. K. R. Vangoor, V. Tarasov, and E. Zadok, "To fuse or not to fuse: Performance of user-space file systems," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017.

[22] gRPC: Google Remote Procedure Call, http://www.grpc.io/.

[23] Google Protocol Buffers, https://developers.google.com/protocol-buffers/.

[24] SQLite Home, https://www.sqlite.org/.

[25] Y. Kim, S. Atchley, G. R. Vallée, and G. M. Shipman, "LADS: Optimizing Data Transfers Using Layout-aware Data Scheduling," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, ser. FAST'15.

[26] IOR: PFS Benchmarking Tool, https://github.com/LLNL/ior.