2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Exploring the Potential of Elastic Computing Clusters in Geo-distributed Data Centers with Fast Fabric Interconnection

Shouwei Chen*†, Wensheng Wang*, Ivan Rodero†
*JD.com Inc., †Rutgers Discovery Informatics Institute

***Abstract*—Large-scale enterprise computing systems are growing rapidly, to address the increasing demand for data processing; however, in many cases, the computing resources in a single data center may not be sufficient for critical data-centric workloads, and important factors, such as space limitations, power availability, or company policies, limit the possibilities of expanding the data center's resources. In this paper, we explore the potential of harvesting spare computing resources across geo-distributed data centers with fast fabric interconnection for real-world enterprise applications. We specifically characterize the computing resource utilization of four large-scale production data centers, and we show how to efficiently combine local storage and computing clusters with remote and elastic computation resources. The primary challenge is incorporating the available remote computing resources efficiently. To achieve this goal, we propose leveraging the capabilities of Kubernetes-based elastic computing clusters to utilize the spare computing resources across geo-distributed data centers for Big Data applications. We also provide an experimental performance evaluation based on real-use case scenarios via an empirical execution and a simulation, which shows that the proposed system can accelerate Big Data services by employing existing computing resources more efficiently across geo-distributed data centers.**

***Index Terms*—Geo-distributed Data Centers, Apache Spark, Kubernetes, Elastic Computing Cluster, Fast Fabric Interconnection.**

I. Introduction

To process large volumes of data, Big Data applications require a large number of computing resources. However, expanding the size of the data center computing clusters within the same geo-location is challenging for large-scale organizations due to several reasons, including space and power supply limitations. For example, we have a computing cluster in Beijing, with about 3,700 servers to support warehouse applications, which has reached the data center's limits. However, these warehouse applications support the data supply for all departments and, as a result, the demand to reduce their execution time is increasing. Furthermore, the cost of expanding the computing capabilities by purchasing more servers can substantially increase the total cost of ownership of enterprise data centers [1], [2], as considerable infrastructure investments might be needed to support additional resources. To reduce the total cost of ownership of enterprise data centers, we harvest spare computing resources from existing computing clusters.

Current research efforts [3]–[7] mostly focus on low-bandwidth cloud environments. The data transmission time for a task across data centers can determine the execution time of applications. In most of these cases, data distribution and caching are reasonable solutions; however, with existing fast fabric interconnections, we integrate computing clusters as an elastic computing resource pool across geo-distributed data centers.

We found the computing resource utilization of different data centers differs in the time scale and the spatial scale. For example, the computing resource utilization of online service clusters (e.g., a search and online shopping service) can be much lower than off-line service clusters during particular periods, because the system has to reserve computing resources for periods requiring peak computing capabilities (e.g., 200% computing resources compared to normal operations). Another difference is in the time scale, as the busy time of online service clusters is typically daytime, while offline service clusters can be busy at that time (e.g., running Extract, Transform, Load workloads). As the resource utilization of offline service clusters is usually high (70% to 90%), running online services simultaneously may require significant additional resources to avoid resource contention and/or workload performance degradation.

With the consideration of network latency and bandwidth as key factors, we explore the use of fast fabric interconnections to overcome this problem. In this paper, we focus on a data warehouse service, which deals with a large volume of data (hundreds of GBs to tens of TBs per application at a PB-level data warehouse). To better understand the performance of different methods with fast fabric interconnection, we first characterize two different situations:

- Performance of data warehouse applications with a storage cluster (HDFS) within the same data center.
- Performance of data warehouse applications with a remote storage cluster (HDFS).

Based on the findings from our empirical performance evaluation, we investigate methods for fully utilizing the computing resources between two geo-distributed data centers with fast fabric interconnection and the abstraction of an elastic computing cluster. To the best of our knowledge, no previous work has explored the execution of enterprise applications with an elastic computing cluster using geo-distributed data centers with fast fabric interconnection. The main contributions of this paper are as follows:

- We characterize the performance of real warehouse applications with fast fabric interconnection between geo-distributed data centers.
- We explore the potential of the elastic computing cluster abstraction with Kubernetes across data centers.
- We explore scheduling strategies for data warehouse applications.
- We provide meaningful evaluations of the proposed approach in real-world large-scale computing cluster deployments.

The rest of the paper is organized as follows. In Section II, we provide background and related work, which motivates the proposed approach. In Section III, we characterize warehouse applications and resource utilization of warehouse applications in a real production environment. The results of the experimental evaluation provide support for harvesting spare computing resources across geo-distributed data centers as this approach can accelerate large-scale Big Data services, such as the evaluated data warehouse service. Finally, in Section IV, we conclude the paper and outline directions for future work.

II. Background and Motivation

In this section, we discuss the main challenges faced in this paper with an analysis of the computing resource utilization of a real production enterprise computing cluster. Then we evaluate the two proposed types of deployment to estimate the performance and cost of network transmission between geo-distributed data centers.

A. Characterizing the resource utilization of computing clusters

We leverage data collection with an internal monitor platform, Big Data Platform Eye (BDPEYE), which is used in production services. With BDPEYE, users can collect different types of metrics from different components (hardware metrics, application metrics, cluster metrics, etc.). We select metrics from schedulers (YARN and K8S) to show the computing resource utilization of different data centers.

BDPEYE collects monitoring information every 30 sec. As part of this work, we collected one month (August 2018) of monitoring information from two different production computing clusters. To show the available computing resource for Spark, we leverage CPU and memory utilization from clusters in Fig. 1. Based on the computing resource utilization of the computing clusters, we classify the potential states of the computing clusters into three categories: overloaded (applications waiting in the queue), healthy (no applications waiting in the queue), and free (the cluster can offer computing resources to other clusters).

Resource limitations in a single data center. With the limited space and power supply in the same geo-location, it is difficult to expand the data center and provide the abstraction of unlimited servers. Because of this reality, in many cases, large production services cannot achieve good performance with limited computing resources. In practice, large Spark applications usually cannot get enough executors (consisting of CPU and memory) on time, which causes a significant performance loss in the production environment. As we can see from the analysis above, we have to find an efficient way to solve these performance challenges with existing computing resources.

B. Harvesting spare computing resources within geo-distributed data centers

Existing work has proposed mechanisms for harvesting spare computing resources within the same data center [8], [9]. As opposed to existing work, and based on the observation from the analysis above, we efficiently utilize computing resources across geo-distributed data centers.

Geo-distributed data centers. Existing work [3], [5]–[7] has explored the potential and methods for utilizing geo-distributed data centers with limited network bandwidth. However, these methods cannot meet the performance requirement for large enterprise data-centric services, which can consist of hundreds or even thousands of applications. This challenge is explained in detail and analyzed in section III.

Building a disaggregated computing resource pool between geo-distributed data centers. As we can see in Fig. 2, four data centers are distributed in four different regions: Beijing Daxing district (DC #1), Beijing Haidian district (DC #2), Beijing Tongzhou district (DC #3), and Langfang city (DC #4).

Table I shows the network bandwidth and straight-line distance between the four data centers. The real network transmission distance is longer than the straight-line distance. Considering the availability of the high bandwidth network, we formulate two design choices of Big Data analytics with geo-distributed data centers. As shown in the table (the capabilities delivered by the data centers with fast and high-capacity fabric interconnections), Big Data analytics has the potential to benefit from additional remote spare computing resources.

TABLE I
Network bandwidth (Gbps)/straight-line distance (KM) between the four geo-distributed data centers

	DC #1	DC #2	DC #3	DC #4
DC #1	- / -	- / 38	- / 33	760 / 37
DC #2	- / 38	- / -	- / 43	760 / 70
DC #3	- / 33	- / 43	- / -	960 / 38
DC #4	760 / 37	760 / 70	960 / 38	- / -

C. Spark-based services on Cloud resources

Currently, cloud computing service provider AWS (Amazon Web Services), Google Cloud and Tencent Cloud provide various Big Data analytics services, including ETL (extract, transform, load), data processing, machine learning, etc. Cloud computing service providers usually provide elastic computing resources with a core storage cluster. Furthermore, some of them also provide auto-scaling services for dynamically increasing storage capabilities (i.e., avoiding data loss). For example, AWS, Google Cloud and Tencent Cloud [10]–[12] provide elastic computing clusters for Spark services and

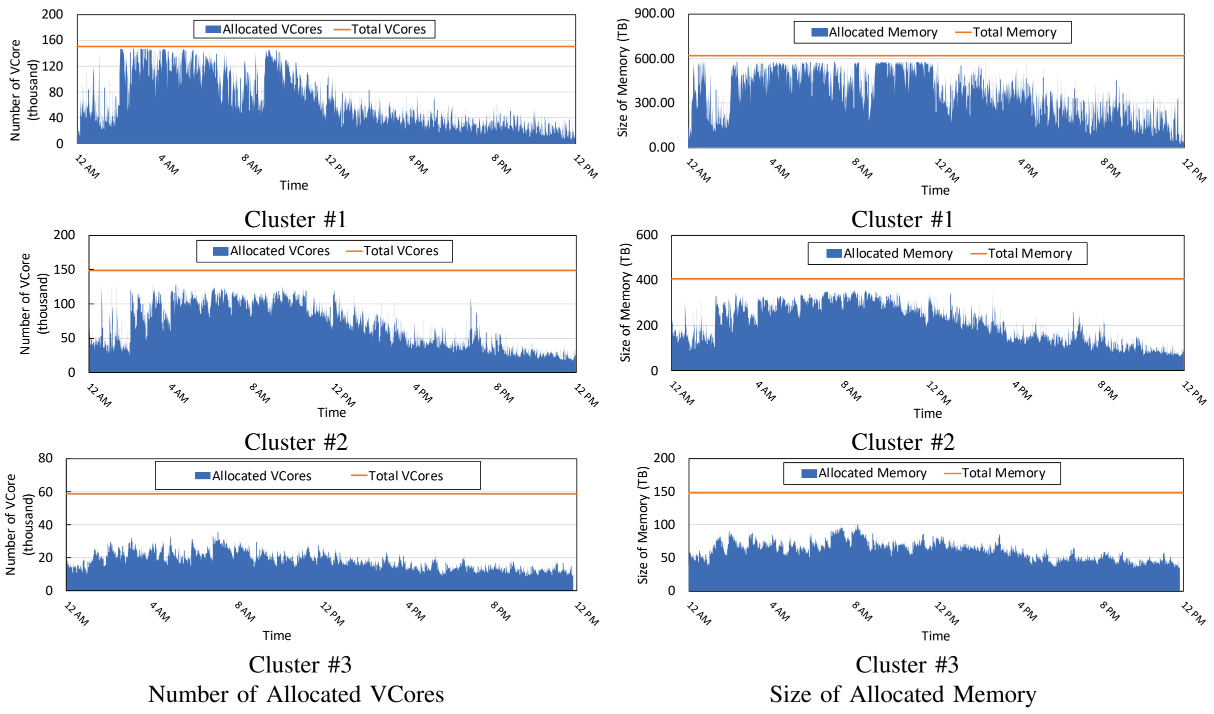

Fig. 1. Vcores and memory utilization of three different computing clusters in geo-distributed data centers

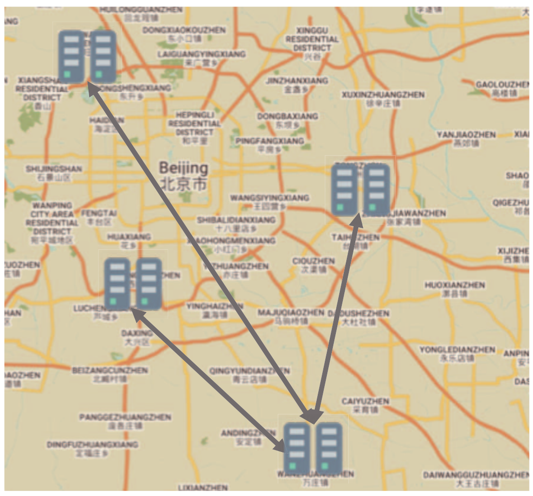

Fig. 2. Geographic locations of the four data centers and the network connection between the data centers. This map is a sketch and does not provide the exact location of the data centers

Google Cloud provides auto-scaling services for the storage layer (HDFS).

In the use case scenario discussed in this paper we have the same Spark service within several different geo-distributed data centers.

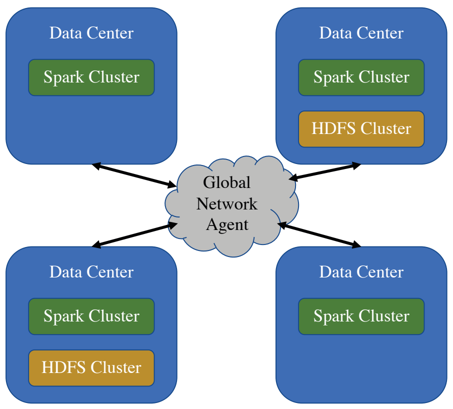

Fig. 3. Architecture of geo-distributed data centers, which is a common architecture in Cloud environments

D. *Modeling the computing cost across data centers*

Because the elastic computing cluster is built with geo-distributed data centers, we have to estimate the cost of network transmission across data centers. Based on our formulation, we can see that it is more realistic for us to use separate computing clusters than one computing cluster. Existing work [4], [13]–[15] proposed network models to calculate the network transmission for Big Data analytics (e.g., MapReduce Hadoop, Spark, Flink, etc.). We adapt the model to fit the targeted elastic cluster model to run with Big Data analytics.

To formulate the cost of network transmission, we use Spark [16] with HDFS [17] as a driving use case. The main costs of data transmission over the network in Spark are read data from HDFS, shuffle, and write the result back to HDFS. Fig. 4 shows the basic data flow for Spark. Spark can implement efficient data locality mechanisms when Spark executors read data blocks from a local node or in a small computing cluster; however, it is harder for Spark to read data blocks from a local node in a large-scale environment. In contrast to the data locality of input data and output data, Spark has good data locality within JVM (memory persistence) and a local disk (disk persistence) for intermediate data. To simplify the formulation, we assume all input and output data is transmitted through the network while intermediate data is not. Thus, the cost of the (network) data transmission ($\mathcal{N}$) can be abstracted as follows:

$$\mathcal{N} = \mathcal{N}_{\text{input}} + \mathcal{N}_{\text{shuffle}} + \mathcal{N}_{\text{output}} \tag{1}$$

Spark assigns a set of operations, which process part of the data within a single process individually. Thus, the Spark performance model can be abstracted as a MapReduce problem. In a traditional MapReduce problem, the performance pattern can be abstracted as a directed acyclic graph $\mathcal{G} \in \{\mathcal{V}, \mathcal{E}\}$.

- Stage is the basic process step in Spark, which is divided by the shuffle operation. $\mathcal{V}$ is a set of stages.
- $\mathcal{E}$ is a set of edges, which represent the execution time of the stages.

Before we formulate the cost of each edge $\mathcal{E}$, we need to elaborate on the shuffling process in Spark. In this paper, we consider the sort-based shuffle in Spark, which is used in the production environment. Spark shuffle can be classified into shuffle write and shuffle read. Shuffle write happens in the "map" phase, while shuffle read happens in the "reduce" phase. Different from MapReduce in Hadoop, Spark does not offer network overlapping transmission for shuffle data. The "reducers" read shuffle data after all "mappers" are finalized in Spark. We can formulate the performance of a single task in a "reducer" as $\mathcal{T}_{\text{task}} = \mathcal{T}_{\text{execution}} + \mathcal{T}_{\text{disk}} + \mathcal{T}_{\text{net}}$ and in "mapper" as $\mathcal{T}_{\text{task}} = \mathcal{T}_{\text{execution}} + \mathcal{T}_{\text{disk}})$.

- $\mathcal{T}_{\text{execution}}$ is the task computing time.
- $\mathcal{T}_{\text{disk}}$ is task disk I/O time, which includes the disk persistence time, shuffle write time in the "mapper" phase, and shuffle read time in the "reducer" phase. We can formulate the disk I/O time for Spark as $\mathcal{T}_{\text{disk}} = \frac{\mathcal{S}_{\text{disk}}}{\mathcal{B}_{\text{disk}}}$. $\mathcal{S}_{\text{disk}}$ is the shuffle data size for a single task, and $\mathcal{B}_{\text{disk}}$ is the disk bandwidth.
- $\mathcal{T}_{\text{net}}$ is the task network transmission time. Furthermore, the main network communication cost are shuffle and fetching input data, and writing output data. To further formulate the network transmission time of shuffle, we use the $\alpha - \beta$ model [18]. Although there are more models, $\alpha - \beta$ model is suitable for the problem that we address. Thus, we can formulate the network transmission time $\mathcal{T}_{\text{net}}$ as $\mathcal{T}_{\text{net}} = \alpha\frac{\mathcal{S}_{\text{net}}}{n} + \frac{\mathcal{S}_{\text{net}}}{\beta}$. α is the latency for sending a message, β is the network bandwidth between two nodes, $\mathcal{S}_{\text{net}}$ is the size of shuffle read, and n is the size of each message.

Thus, we can formulate the execution time ($\mathcal{T}_{\text{application}}$) for Spark as follows:

$$\begin{aligned}\mathcal{T}_{\text{application}} &= \sum\nolimits_{i=0}^{n} \mathcal{T}_{\text{stage}} = \sum\nolimits_{i=0}^{n} max(\mathcal{T}_{\text{task}}) \\ &= \sum\nolimits_{i=0}^{n} max(\mathcal{T}_{\text{execution}} + \mathcal{T}_{\text{disk}} + \mathcal{T}_{\text{net}}) \\ = \sum\nolimits_{i=0}^{n} & max(\mathcal{T}_{\text{execution}} + \frac{\mathcal{S}_{\text{disk}}}{\mathcal{B}_{\text{disk}}} + \alpha\frac{\mathcal{S}_{\text{net}}}{n} + \frac{\mathcal{S}_{\text{net}}}{\beta})\end{aligned} \tag{2}$$

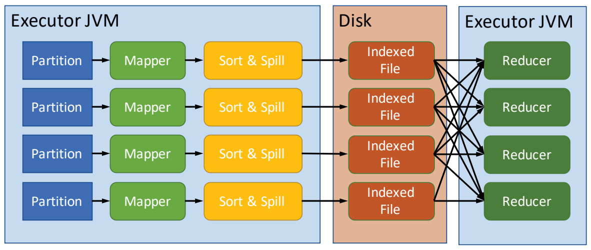

Fig. 4. Spark data flow

To calculate the cost of network transmission intra-data centers and across data centers, we adopt the equations (1) and (2) in the targeted geo-distributed computing cluster model. The total network transmission ($\mathcal{N}$) can be divided into intra-data center transmissions ($\mathcal{N}_{\text{intra}}$) and across-data centers transmissions ($\mathcal{N}_{\text{across}}$). Currently, we use Kubernetes [8], [19] as the primary scheduler for Spark. In this paper, we address two types of deployment to implement the elastic computing cluster for Spark, as shown in Fig. 3.

Scheduling applications within a single computing pool and individual computing pools. We use the driving example to illustrate the cost of network data transmission between geo-distributed data centers for a single Spark application. For the single computing pool condition, Spark gets the available executors, the basic computing unit of Spark, from all four geo-distributed data centers. We assume Spark gets the same number of executors from n data centers in this scenario. For the individual computing pool condition, Spark gets all input data from the local data center or a single remote data center. Thus, we can calculate the cost of network data transmission across data centers for a single computing pool as follows:

$$\mathcal{N}_{\text{across}} = \frac{\text{n-1}}{\text{n}}(\mathcal{N}_{\text{input}} + \mathcal{N}_{\text{output}} + \frac{\text{n-1}}{\text{n}}\mathcal{N}_{\text{shuffle}}) \tag{3}$$

We calculate the cost of network data transmission across data centers for multiple individual computing pools as follows:

$$\mathcal{N}_{\text{across}} = \frac{\text{n-1}}{\text{n}}(\mathcal{N}_{\text{input}} + \mathcal{N}_{\text{output}}) \tag{4}$$

For example, warehouse application #6 is a Spark SQL job with 2,666 GB input data, 3,714 GB shuffle data, and 16 GB output data. We assume that this warehouse application runs on a 300-node computing cluster. The volume of network data transmission across the data centers is $\frac{299}{300} \times \frac{3}{4} \times 3714 + \frac{3}{4} \times (2666 + 16) = 4789GB$ for single computing pools, and

$\frac{299}{300} \times 2666 = 2657GB$ for multiple individual computing pools. The latency between the data centers is longer than the latency within a data center, and the shuffle in Spark could be small all-to-all data communication. The IOPS for shuffle is usually very large [20], which results in high overhead for Spark shuffle data transmission. This calculation does not consider the network transmission, because of the delay scheduler in Spark [21], and metadata transmission with the master node. Note that when this is taken into account, the network transmission could be even higher for a single computing cluster environment.

We adopt the performance equation (2) for geo-distributed data centers. Because of the hardware performance gap between different data centers, Spark can have a significant performance gap or imbalance for different tasks in the same stage. Further, the data skew phenomena are typically found in large-scale enterprise computing clusters with real production data [22]. Scheduling a heavy task in Spark in a low-performance node could cause a significant execution imbalance between different tasks in the same stage.

Based on the analysis above, we can characterize the features of geo-distributed data centers as follows:

- The hardware performance in different data centers can have significant differences. For example, because a new data center may use a newer and better-performing CPU and memory with larger network bandwidth, the performance of Spark in the new data center (DC #4) can save in practice 30% to 50% execution time compared to the old data center (DC #3), depending on the application.
- The data source of the applications is usually in a single data center. Because the volume of the existing data source is huge, it is very challenging to make a copy in each data center.
- The total network bandwidth between data centers is smaller than the total bandwidth within the data center. With fast fabric interconnection between data centers, we can afford part of the network transmission in Spark across geo-distributed data centers.

Long tail problem. In practice, if we have a Spark application running in several data centers, the Spark application could suffer the long tail problem because of hardware heterogeneity. As we can see from equation (2), the execution time $\mathcal{T}_{\text{execution}}$ strongly depends on the performance of the CPU. However, the variability in the CPU performance of servers in different data centers can be significant. Thus, the long tail problem can lead to a large waste of computer resources. As a result, we try to schedule the application within the same geolocation rather than across geo-distributed data centers, when possible.

III. Experimental Evaluation

In this section, we characterize the core services of the data warehouse application running in the production environment. Then we use the data warehouse application workloads to evaluate the performance of the system, locally and with a configuration that uses remote resources.

A. Characterizing data warehouse applications

Data warehouse applications provide data services to various businesses, which include user information, order information, shipping information, storage information, etc. In the time scale, the core service of the data warehouse processes all historical data and provides the data for second-day services. In JD, hundreds of PBs of data are stored in an HDFS cluster, and the data warehouse service has to process about a PB of data every day. For large-scale warehouse applications, tens and even hundreds of data requests are fetched from the distributed file system, and tens of TBs of shuffle data are generated during runtime. Thus, the computing cluster has to provide a large volume of computing resources to allow Spark to provide a reasonable performance.

The services in the enterprise generate an incredibly large volume of data every day. Therefore, the data is stored in a high-compression format to save storage space. Furthermore, there are thousands of input tables, and some of the applications are dependent, so the whole warehouse chain can be divided into seven layers. Figure 5 shows the dependency relationship of the warehouse applications.

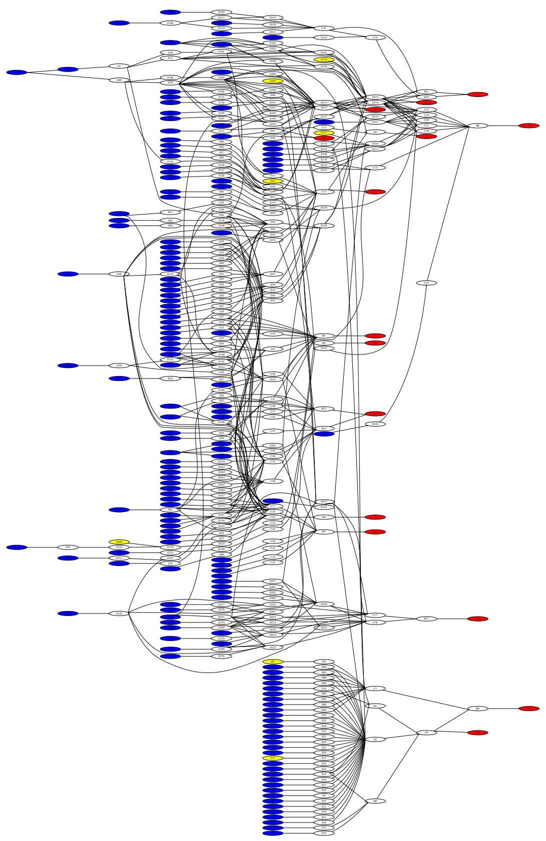

Fig. 5. Dependency relationship between the services of the data warehouse use case. This dependency graph has 435 warehouse applications (143 core data warehouse applications), including basic data movement and a data checker, among others

Fig. 5 shows the relationship between all warehouse applications, including data processing, data movement, data scanning, etc. In the experimental evaluation, we selected the core services of the data warehouse as the target workloads.

The core services of the data warehouse involve a large number of applications, and the data warehouse applications involve a large volume of data that varies in size. Figure 6 shows the input, output, and shuffle(r/w) size of 20 (out of 143) data warehouse applications from the enterprise production environment.

We observe that the core data service of the data warehouse has the following requirements and characteristics:

- High-performance requirements. Because the data service provides data support for other services, the service must be finished as soon as possible.
- Large volume of data with a high-compression format. With the incredible increasing volume of data, high-compression formats can help distributed file systems provide sufficient storage space.
- Large number of applications with several dependent layers. The data warehouse application has pre-requested data warehouse applications in the data pipeline. Thus, the data warehouse application in the next layer cannot start before all applications from the previous layer have finished.
- The input data varies in size. The input data size of the warehouse applications can be significantly diverse, which can cause a big imbalance in the required computing resources.

Resource limitation in Spark applications Currently, the core service of the data warehouse runs on a computing cluster with about 3,700 compute nodes, and includes 143 Spark SQL applications. However, it is hard for each data warehouse application to obtain sufficient computing resources on time, especially applications that require a large volume of data (i.e., require more computing resources to deliver a reasonable performance). In practice, although we use 1,000 executors, sometimes, the application can get only about 400 executors from the computing cluster.

The executor is the minimum computing resource unit, which is composed of virtual cores and memory, in Spark. Currently, each unit has the same number of virtual cores and memory; therefore, we can calculate the number of available executors for certain application as follows:

$$Executor_{available} = Min(\frac{Memory_{available}}{Memory_{executor}}, \frac{VCore_{available}}{VCore_{executor}})$$

B. Testbed

Our testbed is composed of 294 nodes in a production computing cluster. The servers have two Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz processors, and the configuration used in this work includes 256 GB DRAM, and 40 GbE network connectivity between each server. We configure the big data framework, container management system, local distributed file system as the baseline with the same configuration as with the remote distributed file system. Spark version 2.3 was deployed using Kubernetes version 1.10, CentOS version 7.5 and HDFS version 2.7. We reserve one core (3.1%) and 2 GB (0.8%) memory for Kubernetes, and three cores (9.4%) and 9 GB (3.5%) memory for the operating system.

C. Workloads

As we discussed in section II, our target is building an elastic computing pool with individual computing clusters. However, in the geo-distributed environment, the location of data storage could be different than the computing resources. We decide to place the Spark application in different data centers without considering data location based on two reasons: (1) It is hard for engineering to predict the available computing resource in data centers, and (2) most of the existing data is located on the single data center. Thus, it is important for us to find a way to use spare geo-distributed computing resource efficiently. Based on this condition, we evaluate the performance of Spark with remote HDFS and local HDFS. Finally, we use two deployed HDFS systems on data centers #3 and #4, individually.

To evaluate the performance gap between the local data source and remote data source, we select six (typical) warehouse applications from the core service of the data warehouse as shown in table II. We use the input data from production HDFS as standard data. Further, table II shows the input and output size of warehouse applications. Since we need to meet SLA (Service Level Agreement) requirements, we use the most performant configuration for warehouse applications. Thus, we allocate enough memory resource to every executor. The number of cores in each executor is based on the best practices in production environments, which can fully use the I/O throughput for both JVM and disk in the same executor. Furthermore, to evaluate the different type of warehouse applications, we choose different types of applications from the core service of data warehouse, including (1) compute-intensive application: CPU bound application, (2) shuffle intensive application: Spark write and read shuffle data into local disk [23], thus shuffle intensive application is network and disk I/O intensive application, and (3) I/O intensive application: disk I/O bound application.

D. Experimental results

We use Spark with local HDFS as a performance baseline. Then we evaluate the performance of Spark with remote HDFS, which has a separate storage cluster and computing cluster. For each test, we run it five times and use the average execution time as a reference for the performance of the warehouse applications using Spark with local HDFS and Spark with remote HDFS.

Fig. 7 shows the normalized execution time of the warehouse applications. We can observe that the execution time of Spark with the local data source and with the remote data source is almost the same.

In a real production environment, to ensure that the execution time for the data warehouse core services is within

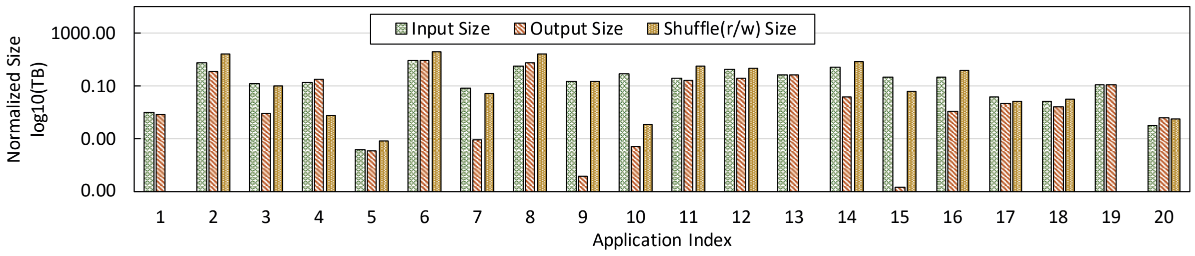

Fig. 6. Input, output and shuffle size of core warehouse applications (20 out of 143)

TABLE II
CHARACTERIZATION AND CONFIGURATION OF WAREHOUSE APPLICATIONS

Application index	Input Size (GB)	Output Size (GB)	Shuffle Size (GB)	Number of Executors	Executor Memory (GB)	Executor Cores
#1	795	2.6	1.00	1000	20	5
#2	165	309	110	1000	20	5
#3	1843	421	2031	1250	24	5
#4	2.85	2.92	2.44	1	20	5
#5	15.1	4.6	7.4	300	20	5
#6	2666	16	3714	1000	20	5

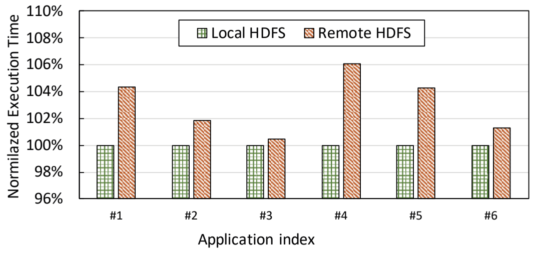

Fig. 7. Normalized execution time of Spark with local HDFS and remote HDFS (with respect to the execution time of Spark with local HDFS)

the SLA constraints, we have to suspend all requests from other queues until the core service has finished. The main reason is that the computing resources in the same geo-location are not enough for such a large service. Thus, we want to harvest the spare computing resource from other data centers. Furthermore, existing research has explored pre-data movement methods based on computing resource prediction. However, because of the growth of and change in existing workloads, the pre-data movement is very challenging to implement in practice. Currently, it is also hard for us to transmit the required data to remote computing clusters based on the computing resource availability before runtime. Thus, transmitting runtime data across data centers is an efficient way to reduce the burden of computing resource utilization in a single computing cluster.

Because we cannot experiment with the performance and resource utilization of the core service of a data warehouse with four data centers, we implement a simulator to reproduce the performance and resource utilization of this system across four geo-distributed data centers. We simulate the execution time for the core services of the data warehouse, and the CPU and memory utilization for four geo-distributed data centers.

In this simulation, based on the existing production environments, we assume that we have one storage cluster in data center 1, and an elastic computing cluster across four different geo-distributed data centers. The performance of a single node in the computing clusters of the various data centers differs. However, to simplify the model, we assume that different computing clusters in different data centers have similar performances.

Data Source: To simulate the overall performance of the elastic computing cluster with the warehouse applications, we use the workloads of the core warehouse applications as shown in Fig. 7. We profile the core utilization, memory utilization, and running time of 143 core warehouse applications in a production environment. The simulation scheduler policy is described as follows:

We mark the co-located computing cluster, which has the same geo-location as a storage cluster, as the highest priority in the scheduling algorithm. It tries to assign applications to cluster 1. Once computing cluster 1 cannot offer enough computing resources to the application, the scheduler assigns remote computing clusters to the application following a round-robin policy. The scheduler then checks the available computing resources in each remote computing cluster before assigning one. Next, the scheduler places the application in the computer cluster that has more available computing resources in different data centers. Further, we divide the core service of the data warehouse into seven layers based on the dependency relationship of their services. Thus, the scheduler cannot start a data warehouse application without finishing the prerequisite data warehouse applications first. Furthermore, we assume that each computing cluster has 10,000 CPU cores with 53 TB of memory. Figure III shows the execution time of the data warehouse core service within a single data center, and with harvesting spare computing resources across geo-distributed data centers, based on the approach described above.

Algorithm 1: Task scheduler policy.

```
1 Function Storage Device Priority ;
  Input : $APP_{pool}$, $App_{index}$, $App_{prerequest}$, $App_{cores}$,
          $App_{memory}$, $App_{time}$, $Cluster_{index}$,
          $Cluster_{index\_cores}$, $Clusters_{index\_memory}$
  Output: $T_{execution}$
  1: start time;
  2: while $APP_{pool}$ is not empty do
  3:   for $i = 1; i <= App_{index}; i++$ do
  4:     if $App_{prerequest} == true$ then
  5:       if $App_{cores} >= Cluster_{1\_cores} \& App_{cores} >= Cluster_{1\_memory}$ then
  6:         Assign Application into cluster 1;
  7:       else if $FindMax(Cluster_{2-4})$ then
  8:         if $App_{cores} >= Cluster_{index\_cores} \& App_{memory} >= Cluster_{index\_memory}$ then
  9:           Assign Application into cluster_index;
 10:         end if
 11:       else
 12:         No cluster available;
 13:       end if
 14:     end if
 15:   end for
 16: end while
 17: end time;
  return ($T_{execution}$);
```

TABLE III
EXECUTION TIME OF CORE SERVICE OF DATA WAREHOUSE

Scenario	Computing cluster	execution time (s)
#1	#1	8,408
#2	#1,#2	5,908
#3	#1,#2,#3	5,272
#4	#1,#2,#3,#4	5,272

IV. Conclusion and Future Work

In this paper, we characterized the computing resource utilization of four different geo-distributed computing clusters. Then we introduced the use of fast fabric interconnections across the geo-distributed data centers. We explored the potential deployment with these data centers. Then, we evaluated the performance of the system based on Spark, HDFS, and Kubenetes in a production enterprise environment. Based on the results, we explored the potential of using fast fabric interconnection to harvest spare computing resources across geo-distributed data centers. We built a simulation based on a data warehouse core service, and then we verified that we could build an elastic computing cluster across geo-distributed data centers, which can speed up large data warehouse enterprise services. Our future work includes exploring in-memory distributed file systems, such as Alluxio, for supporting elastic computing clusters across geo-distributed data centers, and exploring co-design aspects related to Big Data frameworks and next-generation network interconnections.

Acknowledgements

We thank JD cloud engineers for their support of this project. This work is supported in part by National Science Foundation via grants OAC 1640834, OAC 1835692, OAC 1826997, and OCE 1745246.

References

[1] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[3] I. Cano, M. Weimer, D. Mahajan, C. Curino, and G. M. Fumarola, "Towards geo-distributed machine learning," *arXiv preprint arXiv:1603.09035*, 2016.

[4] A. C. Zhou, Y. Gong, B. He, and J. Zhai, "Efficient process mapping in geo-distributed cloud data centers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 16.

[5] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 111–124.

[6] A. Rajabi, H. R. Faragardi, and T. Nolte, "An efficient scheduling of hpc applications on geographically distributed cloud data centers," in *International Symposium on Computer Networks and Distributed Systems*. Springer, 2013, pp. 155–167.

[7] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries." in *OSDI*, vol. 16, 2016, pp. 435–450.

[8] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[9] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, Í. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, no. EPFL-CONF-224446, 2016, pp. 755–770.

[10] "Apache spark on amazon emr," https://aws.amazon.com/emr/features/spark/, accessed: 2018-12-3.

[11] "Scaling clusters," https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/scaling-clusters, accessed: 2018-12-3.

[12] "Sparking on tencent cloud," https://cloud.tencent.com/product/sparkling, accessed: 2018-12-3.

[13] T. Hoefler, E. Jeannot, and G. Mercier, "An overview of process mapping techniques and algorithms in high-performance computing," 2014.

[14] C.-H. Lee, M. Kim, and C.-I. Park, "An efficient k-way graph partitioning algorithm for task allocation in parallel computing systems," in *Systems Integration, 1990. Systems Integration'90., Proceedings of the First International Conference on*. IEEE, 1990, pp. 748–751.

[15] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, "Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 353–360.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. Ieee, 2010, pp. 1–10.

[18] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[19] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, p. 10, 2016.

[20] H. Zhang, B. Cho, E. Seyfe, A. Ching, and M. J. Freedman, "Riffle: optimized shuffle service for large-scale data analytics," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 43.

[21] "Spark data locality," https://spark.apache.org/docs/latest/tuning.html\#data-locality, accessed: 2018-12-3.

[22] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *ACM SIGMOD International Conference on Management of data*, 2013, pp. 13–24.

[23] "Shuffle operations," https://spark.apache.org/docs/latest/rdd-programming-guide.html\#shuffle-operations/, accessed: 2018-12-3.