HDM-MC in-Action: A Framework for Big Data Analytics across Multiple Clusters

Conference Paper · July 2018

DOI: 10.1109/ICDCS.2018.00165

5 authors, including:

Dongyao Wu
UNSW Sydney
14 PUBLICATIONS **148** CITATIONS
SEE PROFILE

Sherif Sakr
King Saud bin Abdulaziz University for Health Sciences
242 PUBLICATIONS **5,488** CITATIONS
SEE PROFILE

Liming Zhu
The Commonwealth Scientific and Industrial Research Organisation
333 PUBLICATIONS **9,485** CITATIONS
SEE PROFILE

HDM-MC in-Action: A Framework for Big Data Analytics across Multiple Clusters

Dongyao Wu*†, Sherif Sakr*‡§, Liming Zhu*†, Sung Une Lee*†, Huijun Wu*†
*Data61, CSIRO, Sydney, Australia
†School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
‡University of Tartu, Tartu, Estonia
§King Saud bin Abdulaziz University for Health Sciences, National Guard, Riyadh, Saudi Arabia
E-mail: {Dongyao.Wu, Sherif.Sakr, Liming.Zhu, Huijun.Wu}@data61.csiro.au

***Abstract*—Big data are increasingly collected and stored in a highly distributed infrastructures due to the development of several emerging technologies including sensor network, cloud computing, IoT and mobile computing among many other emerging technologies. In practice, the majority of existing big-data-processing frameworks (e.g., Hadoop, Spark, Flink) are designed based on the single-cluster setup with the assumptions of centralized management and homogeneous connectivity which makes them sub-optimal and sometimes infeasible to be applied for scenarios that require implementing data analytics jobs on highly distributed data sets (across racks, data centers or multi-organizations). We demonstrate HDM-MC, a big data processing framework that is designed to enable the capability of performing large scale data analytics across multi-clusters with minimum extra overhead due to additional scheduling requirements. We describe the architecture and realization of the system using a step-by-step example scenario.**

I. Introduction

Data processing and analysis technologies have entered the era of Big Data. At the same time, data is increasingly distributed among more complicated infrastructures and network environments due to the development of emerging technologies such as cloud computing, mobile computing and the Internet of Things (IoT) [1]. Cloud computing infrastructures have been witnessing significant growth. Cloud providers such as AWS and Azure have dozens of data centers all over the world. In mobile computing and IoT environments, data are collected and stored in a highly distributed network and infrastructure. Moreover, since data has become a critical resource of modern companies and organizations in almost any domain, more and more companies/organizations have built their own clusters and data centers in order to manage and analyze the user generated data to learn valuable insights. In many scenarios, each of these organizations preserves parts of the features for the same entities due to the diversified products they provide. For example, in current capital companies, they normally have multiple products each of which may have their databases and data repositories. Although, different products can share the same physical infrastructure for their data analytics clusters, different products still prefer to separate their own data sets and management of their computing resources because each product may have different data sensitivity and computation priority considerations. In this case, many large companies logically divide their data processing infrastructures for different product groups. In addition, for global companies such as Google, Facebook and Amazon, they have customers all over the world. Therefore, multiple data centers are setup geographically in order to provide low-latency services to users in different regions. Respectively, there are multiple data processing clusters being constructed in each of these data centers. It is a great challenge for data scientists to apply data analytics over the data across all the data centers as there is very limited network connectivity across those geo-distributed clusters.

Recently, there are increasing requirements for performing data analysis across clusters, data centers or organizations in order to discover more significant insights and knowledge for both data scientists and end users. However, the current state of the art of big data processing frameworks such as MapReduce [2], Spark [3] and Flink [4]) are originally designed for a single cluster architecture, in which the whole cluster is managed by a centralized master and they assume that workers in the cluster are symmetrically connected. Those assumptions do not fit into many scenarios in which data are collected and stored in more widely distributed environments where there could be either physical or logical boundaries for various groups of computational nodes. We demonstrate HDM-MC [5][1], a big data processing framework which is designed to enable the capability of performing large scale data analytics across multi-clusters. HDM-MC applies a set of optimizations that have been introduced to improve the performance for multi-clustered applications. In particular, it uses two multi-cluster architectures (P2P and Hierarchical) which are designed to support managing and coordinating resources (workers) on multi-clusters infrastructures. In addition, it applies a two-layer job planning and scheduling mechanisms which are designed to support the coordination and execution of big data jobs across multi-cluster infrastructures.

II. Background - HDM

HDM-MC [5] is built on a data processing engine – HDM[2], (Hierarchically Distributed Matrix) [6], a big data processing framework with built-in data optimizations for execution and

[1]A demonstration is available on https://youtu.be/ynE2mrwGQls
[2]https://github.com/tiantianwdy/hdm

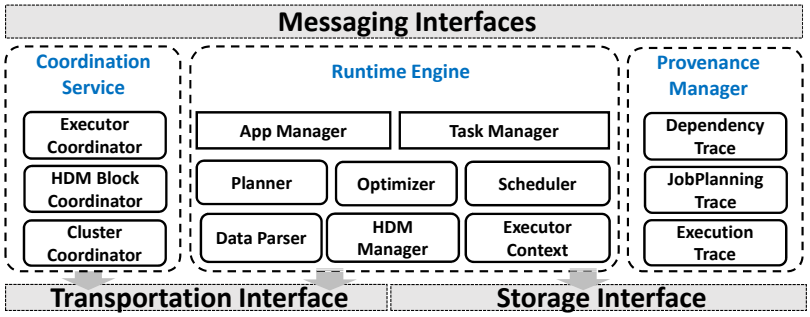

Fig. 1. System Architecture of HDM-MC Framework.

data provenance support for managing continuously evolving big data applications. In particular, HDM is a lightweight, functional and strongly-typed data representation which contains complete information (such as data format, locations, dependencies and functions between input and output) to support parallel execution of data-driven applications [7], [6]. Exploiting the functional nature of HDM enables deployed applications of HDM to be natively integrable and reusable by other programs and applications. In addition, by analyzing the execution graph and functional semantics of HDMs, multiple optimizations are provided to automatically improve the execution performance of HDM data flows. Moreover, by drawing on the comprehensive information maintained by HDM graphs, the runtime execution engine of HDM is also able to provide provenance and history management for submitted applications.

III. System Overview

A. HDM-MC

HDM [6] framework has been initially designed for a single-cluster architecture. To enable a multi-cluster architecture, HDM-MC [5] has extended the cluster coordination, job planning and scheduling components of the kernel of HDM. The extended multi-cluster data processing framework is named as HDM-MC [5]. Figure 1 illustrates the system architecture of the HDM-MC framework.

1) Cluster Coordination: HDM-MC uses two distributed architectures for the coordination of multiple clusters: the P2P architecture and the hierarchical architecture. The P2P architecture in HDM-MC considers the master in each cluster as being equally important as the others. Each master can collaborate with the others to accomplish a cross cluster job by outputting the necessary tasks to the collaborating clusters. For the hierarchical architecture, it assumes that there is a superior master which can communicate with all the other masters such that it can explain and schedule jobs across all the clusters. The P2P architecture is more suitable for cross-organization clusters which do not expect a third-party coordinator. In comparison, the hierarchical architecture is more suitable for the multi-clusters within the same organization, in which a global master can be used to manage and maintain the information for the overall resources.

2) Job Planning: In order to enable the planning for multi-cluster jobs, HDM-MC divides a cross-cluster job into multiple sub-jobs (stages) and categorizes them based on where the sub-job should be executed:

- *Local Job*: If all the input data of a sub-job is in the current cluster, then it can be simply scheduled for execution in the local cluster.
- *Remote Job*: If all the input data of a sub-job is in a remote cluster, then it needs to be output to the remote cluster for execution.
- *Collaborative Job*: If the input data of a sub-job is distributed among multiple clusters, then the job needs to be accomplished by collaborative coordination and execution among the related clusters.

As a result, the job planning process of HDM-MC consists of two steps: the Stage Planning step and the task Planning step. The former is responsible to identify and divide the overall job into different sub-jobs (stages). The latter is responsible for local job explanation just like the a single-cluster planner does.

3) Scheduling: With respect to the job planning phase, the scheduling phase is managed by a two-layer scheduler:

- The first layer is used to monitor and schedule the stages of each application. Once all the parent jobs of a stage are completed, the stage is notified as active and will be submitted to a remote or local task scheduler for execution.
- The second layer is to receive, monitor and schedule the tasks of each active stage. In principle, any classic single cluster scheduler can be applied to this layer.

In the second layer of scheduler, tasks can be scheduled in each local cluster using three types of scheduling strategies: *Delay Scheduling*, *Minmin/Maxmin Scheduling* and *Hungarian Algorithm* [5].

B. Extension for Vectorized Programming

To facilitate the data analytics applications, we extended the core APIs provided by HDM with vectorized abstraction and operations. The data abstraction of vectorized APIs include following data structures:

1) DistributedVector: DistributedVector is a high level interface for the vectors which are implemented in a distributed

manner, it can be used just as a normal vector, but during execution, it outsources the computation to a back-end cluster and then collect the results after accomplishing the computation.

- *DistributedDenseVector*: It is an indexed array of numeric values that been partitioned and stored on a set of distributed nodes. By default the vector is partitioned based on the range of indexes.
- *DistributedSparseVector*: It is a set of tuples that each represents a non-zero value of the vector. Every tuple is formed in a pair of (index, value) for a non-zero element in the vector. The whole vector has been partitioned and stored on a set of distributed nodes. By default the vector is partitioned based on the indexes of the elements.

2) DistributedMatrix: DistributedMatrix is a high level interface for the matrices that are implemented in specific types of distributed abstraction (row-based, column-based, sparse and block-based, etc.) Current concrete realizations of matrix include following types:

- *DistributedRowMatrix*: The matrix is split by rows, each partition of the matrix contains a set of rows and their row indexes.
- *DistributedColumnMatrix*: The matrix is split by columns, whereby each partition of the matrix contains a set of columns and their column indexes.
- *DistributedBlockMatrix*: The matrix is split into a set of blocks. Each block is a sub-matrix block with the offsets of the coordinate position.
- *DistributedSparseMatrix*: The matrix is a collection of points each of which is a non-zero element in the matrix with its row and column indexes.
- *DistributedVerticalBlockMatrix*: The matrix is a high order matrix abstraction, which is composed by a set of vertically partitioned sub-matrix blocks. Each sub-matrix can be a basic type distributed matrix.
- *DistributedHorizontalBlockMatrix*:The matrix is a high order matrix abstraction, while it is composed by a set of horizontally partitioned sub-matrix blocks.

When creating a matrix abstraction in multi-cluster scenarios, the matrix can be either vertically partitioned (DistributedColumnMatrix and DistributedVerticalBlockMatrix) or horizontally partitioned (DistributedRowMatrix and DistributedHorizontalBlockMatrix) based on the data distribution logics.

C. Security, Privacy and Reliability Concerns

Security and privacy are important factors to be concerned for multi-cluster computation. HDM-MC provides basic mechanisms such as encryption as a option config in case that users want to ensure the data privacy during data transferring process across clusters. It also allows workers among different security domains to create secure connections instead of normal TCP connections. As transportation across different clusters could be unstable, HDM-MC also supports reconnecting and backup of replications during data transferring. When a worker is temporarily unreachable, the connecter can try to reconnect to the worker until it meets the maximum number of failures. If a worker fails for a long time, the connector will try to connect to a backup node to send and transmit data.

IV. Demonstration

A. Scenario Overview

In our demonstration, we setup two clusters in different availability zones in AWS. In each cluster, we deploy HDFS for data persistence and HDM framework for distributed data processing across clusters. The two HDM masters in the two clusters communicate to each other based on the P2P coordination protocol. Each of the cluster consists of 10 M3.xlarge nodes for computation. The data set in each cluster is 61.3GB (1 billion records with 10 columns for each record).

In the demonstration scenario, we conduct the SGD-based logistic regression algorithm written in HDM (as listed in Listing 1) and submit to one of the cluster master to trigger the execution. In this scenario, there are two organizations, each of which holds parts of the features that are used for the training process. In addition, each of the organizations has its own data processing clusters while they do not want to directly share the raw data to each other. In order to learn a complete model or insight from the full data set shared among the two organizations, there is a need for a data processing framework which can collaboratively accomplish distributed processing across the two (or multiple) clusters. The algorithm is set with a maximum of 3 iterations for the demonstration. And after the training is finished in each iteration, the client-side program can collect the updated co-efficiency and output in the console.

```
val m1 = HDMatrix.csv("hdfs://10.10.0.100:9091/user/data1")
val m2 = HDMatrix.csv("hdfs://10.20.0.100:9091/user/data2")
//join as training data
val x = HDMatrix.vertical(m1, m2)
val y = HDVector.csv("hdfs://10.10.0.100:9091/user/labels")
val weights = Vector.init(Math.Random() * 0.001)
val numIteration = 5
for (i <- 1 to numIteration){
    val gradient = (y - x.dot(weights).sigmoid()) *: x
    weights -= gradient
}
println(weights)
```

Listing 1. Code Snippet of SGD Logistic Regression across two Clusters

B. Demo steps

The demonstration of the HDM-MC framework will go through four main steps.

1) Multi-cluster Resource Monitoring: In this step of the demonstration, we will show the multi-cluster resource manager (Fig. 2). In the resource manager console, we can see that the manager maintains the topology of the clusters including both masters and workers. In addition, each cluster master supervises and monitors the states and resources of its children workers. The collaborating masters are aware of the overall resources of each other but the detailed information of local workers is transparent to remote masters. For each worker, the master is able to monitor the CPU, Memory, Network and JVM utilization of every worker and maintain the information

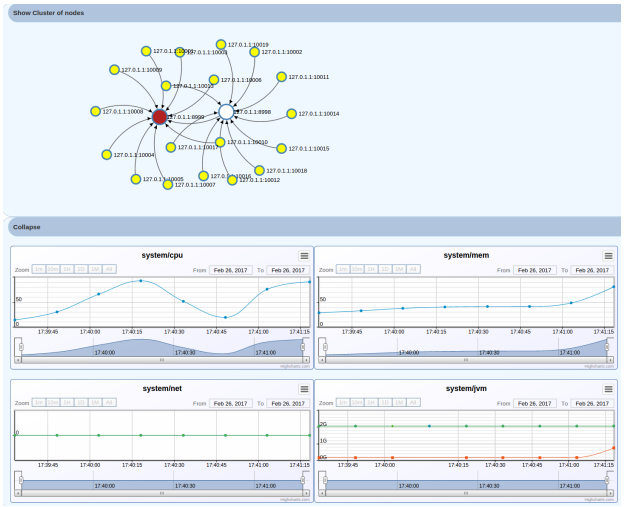

Fig. 2. Resource Management of Multiple Clusters in HDM-MC.

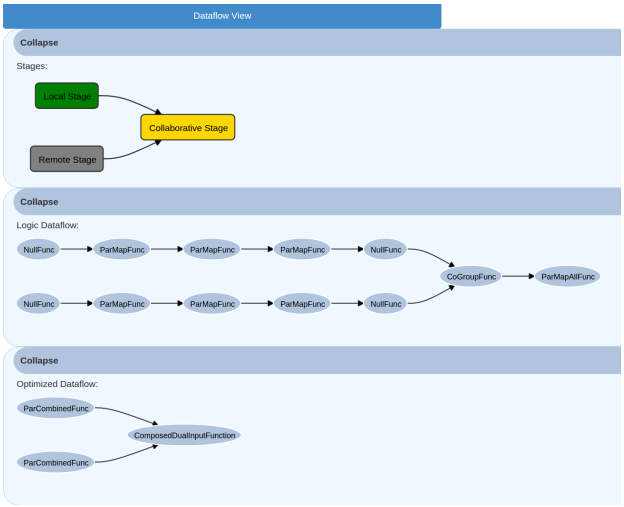

Fig. 3. Job Planning of HDM-MC Applications.

for certain time duration. The resource information maintained in the cluster topology is used for the future job planning, scheduling and execution.

2) Data Distribution among Multiple Clusters: We will present the data block distribution information after specify the matrix abstraction among the two multi-clusters. The system console of HDM-MC visualizes the data block distributions with the cluster boundary and meta-data of each blocks. The data distribution visualization gives a user a easy view and understanding of the underlying locations of data blocks and how they are logically organized.

3) Multi-cluster Job Planning: We will present the job planning of a cross cluster application (Fig. 3). In particular, we use the SGD-LR performed across the two clusters as the example. For each iteration of the job it is divided into three stages:

- Local jobs which load the data in parallel and perform local computation and aggregation on each chunk of the data sets.
- Remote jobs that load remote data and perform cluster-local computation in the remote cluster.
- Collaborative jobs that perform the stochastic gradient process among the data blocks across the cluster.

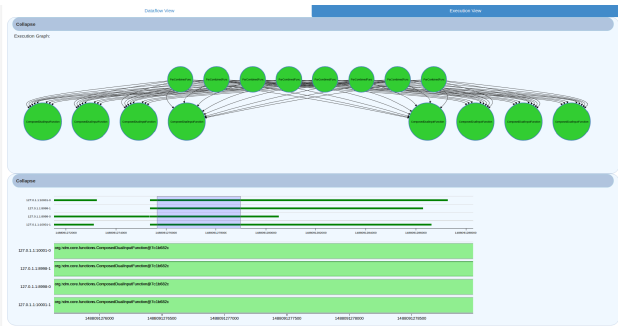

Fig. 4. Scheduling and Execution of HDM-MC Applications.

4) Multi-cluster Scheduling and Execution: We will present the scheduling and execution process (Fig. 4) for the cross-cluster logistic regression. With respect to the planning phase, the scheduling process of SGD-LR is divided into two-layers:

- *Stage Scheduling*: The first layer is stage scheduling. The SGD-LR consists of three stages: a local stage for parallel execution in the current cluster, a remote stage for parallel execution in another cluster and the last stage that collaboratively aggregates the gradients learned in each data chunk.
- *Task Scheduling*: In the task scheduling layer, the tasks of the first two stages are scheduled and executed as in the single cluster environment. However, in the collaborative stage, tasks may require some parts of data from a remote cluster. In this case, a remote cluster URL is used to represent and request a partition of data. Cross cluster data communication could be encrypted and decrypted based on a security agreement (e.g a public-private key pair) of the two clusters.

In addition to the HDM-MC framework demonstration, we will also discuss in more details about the design choices that we have made on designing the various components of the framework. In addition, performance comparison with the Spark framework [3], using different operations, will be presented to demonstrate the effectiveness of the HDM-MC optimization techniques.

REFERENCES

[1] S. Sakr, *Big Data 2.0 Processing Systems - A Survey*, ser. Springer Briefs in Computer Science. Springer, 2016.
[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, 2008.
[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *HotCloud*, 2010.
[4] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink™: Stream and Batch Processing in a Single Engine," *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, 2015.
[5] D. Wu, S. Sakr, L. Zhu, and H. Wu, "Towards Big Data Analytics across Multiple Clusters," in *Cluster, Cloud and Grid Computing (CCGrid), 17th IEEE/ACM International Symposium on*. IEEE/ACM, 2017.
[6] D. Wu, L. Zhu, Q. Lu, and S. Sakr, "HDM: A Composable Framework for Big Data Processing," *IEEE Transactions on Big Data*, 2017.
[7] D. Wu, S. Sakr, L. Zhu, and Q. Lu, "Composable and efficient functional big data processing framework," in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 279–286.