International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 3

ISSN: 2454-4248
646 – 649

# Analysis of Big Data Processing Using HDM Framework

Mr.Rajat Bodankar[1], Ms.Roshani Talmale[2], Mr.Rajesh Babu[3]

[1st]M.Tech Student 2nd Year Dept. of Computer Science and Engineering Tulsiramji Gaikwad-Patil College of Engineering and Technology Nagpur, India

*E-mail :rbodankar@gmail.com* .

[2nd] Project Guide Dept. of Computer Science and Engineering Tulsiramji Gaikwad-Patil College of Engineering and Technology Nagpur, India

[3nd] Project Co-Guide Dept. of Computer Science and Engineering Tulsiramji Gaikwad-Patil College of Engineering and Technology Nagpur, India

***Abstract:*** MapReduce and Spark have been introduced to ease the task of developing big data programs and applications. However, the jobs in these frameworks are roughly defined and packaged as executable jars without any functionality being exposed or described. This means that deployed jobs are not natively composable and reusable for subsequent development. Besides, it also hampers the ability for applying optimizations on the data flow of job sequences and pipelines. The Hierarchically Distributed Data Matrix (HDM) which is a functional, strongly-typed data representation for writing composable big data applications. Along with HDM, a runtime framework is provided to support the execution, integration and management of HDM applications on distributed infrastructures. Based on the functional data dependency graph of HDM, multiple optimizations are applied to improve the performance of executing HDM jobs. The experimental results show that our optimizations can achieve improvements between 10% to 30% of the Job-Completion-Time and clustering time for different types of applications when compared.

***Keywords:****Big data processing, parallel programming, functional programming, system architecture*

*****

## Introduction

Big data has become a popular term which is used to describe the exponential growth and availability of data.the growing demand for large-scale data processing and data analysis applications spurred the development of novel solutions to tackle this challenge [10]. For about a decade,the mapreduce framework has represented the defacto standard of big data technologies and has been widely utilized as a popular mechanism to harness the power of large clusters of computers. In general, the fundamental principle of the mapreduce framework is to move analysis to the data, rather than moving the data to a system that can analyze it. It allows programmers to think in a data-centric fashion where they can focus on applying transformations to sets of data records while the details of distributed execution and fault tolerance are transparently managed by theFramework. However, in recent years, with the increasing applications' requirements in the data analytics domain, various limitations of the hadoop framework have been recognized and thus we have witnessed an unprecedented interest to tackle these challenges with new solutions which constituted a new wave of mostly domain-specific, optimized big data processing platforms.

Big Data is the large and complex data that is difficult to use the traditional tools to store, manage, and analyze in an acceptable duration. Therefore, the Big Data needs a new processing model which has the better storage, decision-making, and analyzing abilities. This is the reason why the Big Data technology was born. The Big Data technology provides a new way to extract, interact, integrate, and analyze of Big Data. The Big Data strategy is aiming at mining the significant valuable data information behind the Big Data by specialized processing. In other words, if comparing the Big Data to an industry, the key of the industry is to create the data value by increasing the processing capacity of the data.Big Data is always online and can be accessed and computed. With the rapid developments of the Internet, the Big Data is not only big but is also getting online. Online data is meaningful when the data connects to the end users or the customers. Taking an example, when users use Internet applications, the users' behavior will be delivered to the developers immediately. These developers will optimize the notifications of the applications by using some methods to analyze the data.

## 2. HDM Framework

### 2.1 Overview

Fig 1 shows the system architecture of the HDM runtime engine which is composed of three main components:

 ISSN: 2454-4248
646 – 649

Runtime Engine: is responsible for the management of HDM jobs such as explaining, optimization, schedul-ing and execution. Within the runtime engine, the AppManager manages the information of all deployed jobs. TaskManager maintains the activated tasks for runtime scheduling in the Schedulers; Planner and Op-timizers interpret and optimize the execution plan of HDMs in the explanation phases; HDM manager man-ages the information and states of the HDM blocks in the entire cluster; Execution Context is an abstraction component to support the execution of scheduled tasks on either local or remote nodes. Coordination Service: is composed of three types of co-ordinations: cluster coordination, block coordination

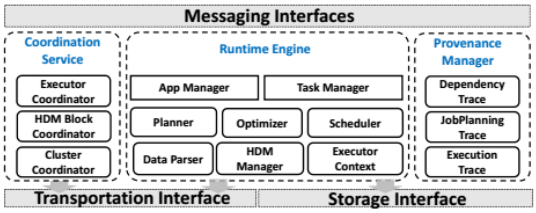


Figure 1: System Architecture of HDM Framework.

and executor coordination. They are responsible for the coordination and management of node resources, distributed HDM data blocks and executors on work-ers, respectively. Data Provenance Manager: is responsible to interact with the HDM runtime engine to collect and main-tain data provenance information (such as Dependen-cyTrace, JobPlanningTrace and ExecutionTrace) for HDM applications. Those information can be queried and obtained by client programs through messages for the usage of analysis or tracing.

### 2.2 HDM Data Flow Optimization

One key feature of HDM is that, the execution engine contains built-in planners and optimizers to automatically optimize the functional data ow of submitted applications and jobs. During explanation of HDM applications, the data ow are represented as DAGs with functional dependencies among operations. The HDM optimizers traverse through the DAG to reconstruct and modify the operations based on optimization rules to obtain more optimal execution plans. Currently, the optimization rules implemented in the HDM optimizers include: function fusion, local aggregation, oper-ation reordering and data caching for iterative jobs [5]. Function fusion. During optimization, the HDM planner combines the lined-up nonshue operations into one operation with high-order function so that the sequence of operations can be compute within one task rather than separate ones to reduce redundant inter-mediate results and task scheduling. This rule can be applied recursively on a sequence of fusible operations to form a compact combined operation. Local Aggregation. Shu e operations are very expensive in the execution of data-intensive applications. If a shue operation is followed with some aggregations, in some cases, the aggregation or part of the aggregation can be applied before the shuing stage. During optimization, HDM planer tries to move those aggregation operations forward before the shuing stage to reduce the amount of data that needs to be transferred during shuing. Operation reordering/reconstruction. Apart from ag-gregations, there are a group of operations which l-ter out a subset of the input during execution. Thoseoperations are called pruning operations1. The HDM planner attempts to lift the priority of the pruning operations while sinking the priority of shu e-intensive operations to reduce the data size that needs to be computed and transferred across the network. Data Caching. For many complicated and pipelined analytics jobs (such as machine learning algorithms), some intermediate results of the job could be reused multiple times by the subsequent operations. There-fore, it is necessary to cache those repetitively used data to avoid redundant computation and communica-tion. In this case, HDM planner counts the reference for the output of each operation in the functional DAG to detect the potential points that intermediate results should be cached for reusing by subsequent operations. During optimization process, the rule above are applied one by one to reconstruct the HDM DAG and the optimiza-tion can last multiple iterations until there is no change in the DAG or it has reached the maximum number of itera-tions. The HDM optimizer is also designed to be extendable by adding new optimization rules by developers when it is needed.

## 3. HDM Programming

One major target of contemporary big data processing frameworks is to ease the complexity for developing data parallel programs and applications. In HDM, functions and operations are defined separately to balance between performance and programming flexibility.

TABLE II. SEMANTICS OF BASIC FUNCTIONS

| Function | Semantics |
|---|---|
| Null | Doing nothing but return input |
| Map<br>$f : T \rightarrow R$ | $F_p$: return $List[T].map(f)$<br>$F_a$: return $List[R]$ += $List[T].map(f)$<br>$F_c$: return $List1[R]$ + $List2[R]$ |
| GroupBy<br>$f : T \rightarrow R$ | $F_p$: return $List[T].map(f)$<br>$F_a$: return $List[R]$ += $List[T].map(f)$<br>$F_c$: return $List_1[R]$ + $List_2[R]$ |
| Reduce<br>$f : T \rightarrow R$ | $F_p$: return $List[T].map(f)$<br>$F_a$: return $List[R]$ += $List[T].map(f)$<br>$F_c$: return $List_1[R]$ + $List_2[R]$ |
| Filter<br>$f : T \rightarrow R$ | $F_p$: return $List[T].map(f)$<br>$F_a$: return $List[R]$ += $List[T].map(f)$<br>$F_c$: return $List_1[R]$ + $List_2[R]$ |

HDM Functions In HDM, a function specifies how input data are transformed as the output. Functions in HDM have different semantics targeting different execution context.

 ISSN: 2454-4248
646 – 649

Basically, one HDM function can have three possible semantics, indicated as Fp, Fa, Fc:

Fp : List[T] → List[R] (1)

Fa : (List[T], List[R]) → List[R] (2)

Fc : (List[R], List[R]) → List[R] (3)

Fp is the basic semantics of a function which specifies how to process one data block. The basic semantics of HDM function assume that the input data is organized as a sequence of records with type T. Similarly, the output of all the functions are also considered as a sequence of records. Based on the type compatibility, multiple functions can be directly pipelined. Fa is the aggregation semantics of a function which specifies how to incrementally aggregate a new input partition to the existing results of this function. Normally, functions are required to be performed on multiple data partitions when the input is too large to fit into one task. The aggregation semantics are very useful under such situations in which accumulative processing could get better performance. Aggregation semantics exist for a function only when it is capable to be represented and calculated in an accumulative manner. Fc is the combination semantics for merging multiple intermediate results from a series of sub-functions to obtain the final global output. It is also a complement for the aggregation semantics when a function is decomposable using the dividecombine pattern. During the explanation of HDM jobs, different semantics are automatically chosen by planers to hide users from functional level optimizations. To better explain the three types of semantics described above, an illustration of the semantics of some basic HDM functions are listed in TABLE II. For the Map function, Fp applies transforming function f to every element then returns the output List with output type R; Fa performs the transformation on every element of the input and then appends the output elements to the existing List; Fc combines the newer output list into the older one. For the GroupBy function, Fp groups the elements according to the mapping function f, then return the list of grouped elements as output; Fa performs mapping function f on every new input element to find the related group, then add the element to the existing group; Fc combines all the groups from the latter list of groups according to the group key.

## 4. Related Work

In principle, the MapReduce framework is originallydesigned to operate on multiple cluster environments. There-fore, it is not well developed to support the executionon highly distributed infrastructures and widely-networkedclusters. To address this issue, many research works haveattempted to extend the MapReduce framework to supporthighly distributed environments such as Grid [2], [11], multi-clusters/clouds[9]. Hierarchical MapReduce framework that introduces global reduce and locality aware scheduling. Theypresent another hierarchical framework [7] which can co-ordinate multiple clusters to run MapReduce jobs amongthem.Hadoop to support schedule data processing on multi-datacenters/clusters and can provide larger pool of processingand data storage. MapReduce framework that can efficiently execute MapRe-duce jobs on geo distributed data sets. However, the approach is highly complex and does notsupport complicated job sequences well. Compared with thisgroup of works which focus on extending MapReduce tosupport highly distributed environment and geo-distributed data sets, HDM provides the capability to explain andschedule general functional data analytics applications onmulti-cluster infrastructures.

## 5.Analysis Of HDM Framework

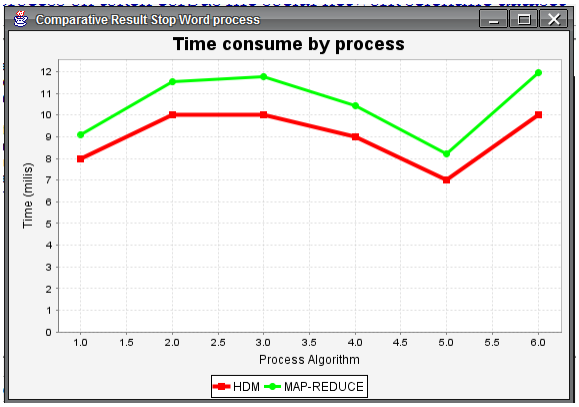


Figure 5.1 Time consume by process on HDM and Map reduce Framework.

Fig.5.1 shown the time consume on text data analysis and result show the compared between map-reduce and HDM framework.

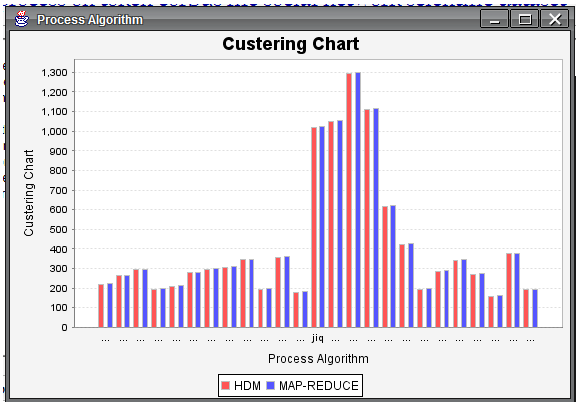


Figure 5.2 Clustering data process on HDM and Map reduce Framework.

Fig 5.2 show clustering process chart on text data and displayed analysis of both framework.

 ISSN: 2454-4248
646 – 649

## 6. Conclusion

In this paper conclude that analysis of HDM framework with big data and compared analysis with Map-Reduce framework.The data flows of HDM jobs are automatically optimized before they are executed in the runtime system. HDM as a functional and strongly-typed meta-data abstraction, along with a runtime system implementation to support the execution, optimization and management of HDM applications.

## Acknowledgement

We take privilege to greet our beloved parents for their encouragement in every effort. Also, we are happy to thank our college management, principal, head of the department and my colleagues for their sincere support in all concerns of resources.

## References

[1]. P. Carbone, A. Katsifodimos, S. Ewen, V. Markl,S.Haridi, and K. Tzoumas. Apache inkTM: Stream and batch processing in a single engine. IEEE Data Eng. Bull., 38(4):28{38, 2015.

[2]. J. Dean and S. Ghemawat. MapReduce: simpli ed data processing on large clusters. Commun. ACM, 51(1), 2008.

[3]. S. Sakr. Big Data 2.0 Processing Systems - A Survey. Springer Briefs in Computer Science. Springer, 2016.

[4]. D. Sculley, G. Holt, D. Golovin, E. Davydov, T.Phillips, D. Ebner, V. Chaudhary, and M. Young. Machine learning: The high interest credit card of technical debt. In SE4ML: Software Engineering for Machine Learning, 2014.

[5]. D. Wu, S. Sakr, L. Zhu, and Q. Lu. Composable and E cient Functional Big Data Processing Framework. In IEEE Big Data, 2015.

[6]. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In HotCloud, 2010.

[7]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributed hadoop mapreduce on the grid. In SC, 2012.

[8]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributed hadoop mapreduce on the grid. In SC, 2012.

[9]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributed hadoop mapreduce on the grid. In SC, 2012.

[10]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributed hadoop mapreduce on the grid. In SC, 2012.

[11]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributed hadoop mapreduce on the grid. In SC, 2012.

[12]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributedhadoop mapreduce on the grid. In SC, 2012.

[13]. C. He, D. Weitzel, D. Swanson, and Y. Lu. Hog: Distributedhadoop mapreduce on the grid. In SC, 2012.

[14]. Y.-L. Su et al. Variable-sized map and locality-aware reduceon public-resource grids. FGCS, 27(6), 2011.

[15]. L. Wang et al. Mapreduce across distributed clusters for data-intensive applications. In IPDPS Workshops, 2012.

[16]. L. Wang et al. G-hadoop: Mapreduce across distributed datacenters for data-intensive computing. FGCS, 29(3), 2013