© 2018 IJSRSET | Volume 4 | Issue 6 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section : Engineering and Technology

Big Data Processing with Data Provenance Using HDM Framework

Rajat Bodankar[1], Roshani Talmale[2]

[1]M.Tech Scholar, Department of Computer Science and Engineering Tulsiramji Gaikwad-Patil College of Engineering and Technology Nagpur, Maharashtra, India

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

ABSTRACT

Big Data applications are becoming more complex and expe-riencing frequent changes and updates. In practice, manual optimization of complex big data jobs is time-consuming and error-prone. Maintenance and management of evolving big data applications is a challenging task as well. We demon-strate HDM, Hierarchically Distributed Data Matrix, as a big data processing framework with built-in data ow op-timizations and integrated maintenance of data provenance information that supports the management of continuously evolving big data applications. In HDM, the data ow of jobs are automatically optimized based on the functional DAG representation to improve the performance during ex-ecution. Additionally, comprehensive meta-data related to explanation, execution and dependency updates of HDM ap-plications are stored and maintained in order to facilitate the debugging, monitoring, tracing and reproducing of HDM jobs and programs.

Keywords : Big Data, Data Flow Optimization, Provenance Management

I. INTRODUCTION

We are experiencing the era of big data that has been fu-elled by the striking speed of the growth in the amount of data that has been generated and consumed. Several big data processing frameworks (e.g., MapReduce [2], Spark [6] and Flink [1], etc.) have been introduced to deal with the challenges of processing the ever larger data sets [3]. These frameworks signi cantly reduce the complexity of writing large scale data-oriented applications. However, in practice, as big data programs and applications have become more and more complicated, it is almost impossible to manually optimize the performance of programs written by diversi ed programmers. Therefore, built-in optimizers are crucial for tackling the challenges of improving the performance of ex-ecuting those hand-written programs and applications. At the same time, realistic data analytics applications are con-tinuously evolving in order to deal with the non-stop changes in the real world. In practice, managing and analyzing those continuously evolving big data applications have resulted in big technical debts [4]. Therefore, there are increasing re-quirements for data provenance to support analyzing, trac-ing and reproduction of historical versions of data analytics applications. In this paper, we demonstrate HDM, (Hierarchically Dis-tributed Matrix) [5], a big data processing framework with built-in data optimizations for execution and data prove-nance supports for managing continuously evolving big data applications. In particular, HDM is a lightweight, functional and strongly-typed data representation which contains com-plete information (such as data format, locations, dependen-cies and functions between input and output) to support parallel execution of data-driven applications [5]. Exploit-ing the functional nature of HDM enables deployed appli-cations 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, mul-tiple optimizations are provided to automatically improve the execution performance of HDM data ows. 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.

II. HDM FRAMEWORK

2.1 System Overview

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

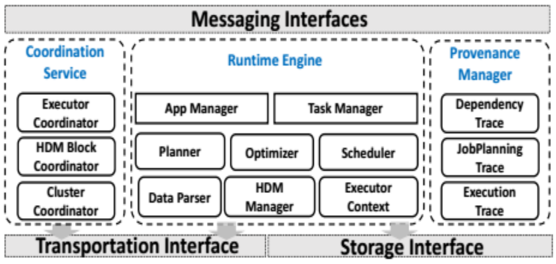

Figure 1: System Architecture of HDM Framework.

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 plan-ner combines the lined-up non-shu e operations into one operation with high-order function so that the se-quence 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 shu e operation is followed with some aggregations, in some cases, the aggregation or part of the aggregation can be applied before the shu ing stage. During optimization, HDM planer tries to move those aggregation operations forward before the shu ing stage to reduce the amount of data that needs to be transferred during shu ing. 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 op-erations 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. Therefore, 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 iterations. The HDM optimizer is also designed to be extendable by adding new optimization rules by developers when it is needed.

2.3 Data Provenance Supports in HDM

It is normally tedious and complicated to maintain and manage applications that are continuously evolving and be-ing updated. In HDM, drawing on comprehensive meta-data information maintained by HDM models, the runtime engine is able to provide data provenance supports includ-ing execution tracing, version control and job replay in the dependency and execution history management component. Basically, the HDM server maintains three types of meta-data about each submitted HDM jobs including Execution-Trace, JobPlanningTrace and DependencyTrace. DependencyTrace. For every submitted HDM program, the server stores and maintains the dependent libraries required for execution. The dependencies and update history are maintained as a tree structure. Based on this information, users are able to reproduce any ver-sion of the submitted applications in the history. JobPlanningTrace. The HDM server also stores the explanation and planning traces for every HDM appli-cations. JobPlanningTrace includes the logical plan, optimizations applied and nal physical execution plan after being parallelized.

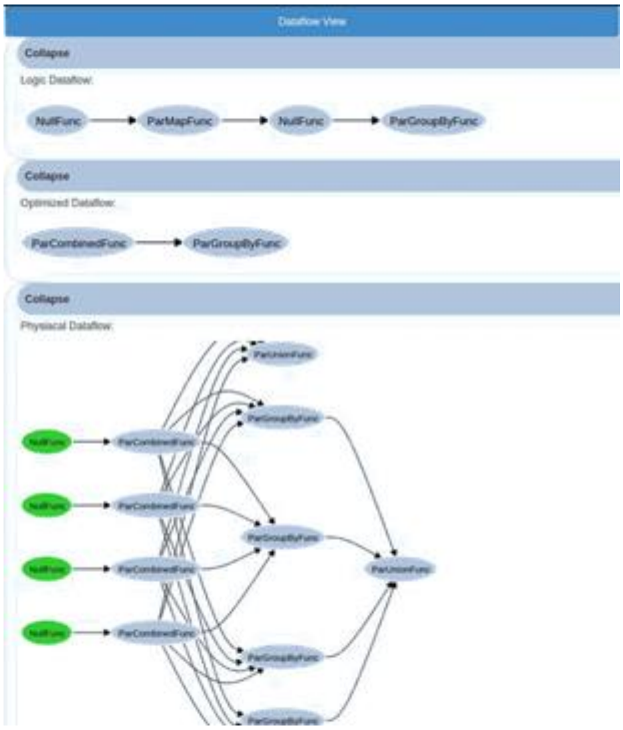

Figure 2: Data ow Visualization of HDM Applica-tions.

ExecutionTrace. During execution, the HDM server also maintains all the runtime information (execution location, input/output, timestamps and execution status, etc.) related to each executed task and job. These information are very meaningful to monitor and trace back the process of execution of historical jobs and applications. Drawing on the three types of information maintained in the HDM server, client-side programs can send messages to query and obtain the history and provenance information, so that users and administrators can pro le, debug and apply analysis to the deployed applications throughout their life cycles.

III. DEMONSTRATION SCENARIOS

In this demonstration, we will present to the audience the HDM framework1 from four main aspects: cluster re-source monitoring, visualisation data ow optimization, exe-cution history tracing, version-control and dependency man-agement. The demonstration will be conducted on AWS EC2 with one M3.Large instance as the master and 10 nodes M3.XLarge instances as the workers. To show how HDM optimizes the data ow and provides data provenance support for its applications, we will present an example of Twitter analysis scenario that consists of the following two Tweets analysis programs

Listing 1: Code Snippet of Finding out Tweets

```
val input = HDM ( " hdfs : / / 1 0 . 1 0 . 0 . 1 0 0 : 9 0 9 1 / user / tweets " ) val
tweets = input . map { line = >
            val seq = line . split ( " , " ) Tweet
            ( seq )
      }
val  grouped  =  tweets . groupBy ( t = >  t . hashTag )
val  results  =  grouped . fin dByK ey ( _ . contains ( " election " ))
```

Listing 2: Code Snippet of Hashtag Counting for Interested Tweets

```
val input = HDM ( " hdfs : / / 1 0 . 1 0 . 0 . 1 0 0 : 9 0 9 1 / user / tweets " ) val
tweets = input . map { line = >
            val seq = line . split ( " , " ) Tweet
            ( seq )
      }
val  grouped  =  tweets . groupBy ( t = >  t . hashTag )
val  trumpN  =  grouped . fin dBy Key ( _  ==  " Trump " ). count
val hillaryN = grouped . find ByK ey ( _ == " Hillary " ). count println ( trumpN
/ hillaryN )
```

Cluster Resource Management. In the rst part of the demo, we will show the cluster resource monitor of the HDM manager. The HDM server maintains the resource-related information of all the workers within the cluster. In the HDMConsole, it is able to monitor the resource utilization information (such as CPU, Memory, Network and JVM) for each worker in real time. Therefore, cluster administrator is able to use these information and easily supervise and understand the status of every worker as well as the entire cluster.

Data ow Optimizations. The second part of the demo shows how the Tweets programs are represented in the HDM DAG and how it is explained, optimized and parallelized by the planner.

For the firrst program, the HDM optimizer applies op-erations reordering to lift the pruning operation find-ByKey to be in front of the shu e operation groupBy. Then the optimizer applies function fusion rule to com-bine map and findBy into a single composite operation.

For the second program, the HDM optimizer applies operation reordering to move the findByKey operation to be in front of groupBy then applies local aggrega-tion count by adding local count in front of groupBy. Lastly, it detects the input tweets that are reused by two operations so that the optimizer can add a cache point after the compute operation that generates the output of tweets.

The HDM server maintains all the related meta-data (such as the creator, original program, logical plan, physical plan, etc.) to all the submitted HDM applications. In the demon-stration, the HDM console visualizes the original logical ow, optimized logical ow and parallelized physical graph

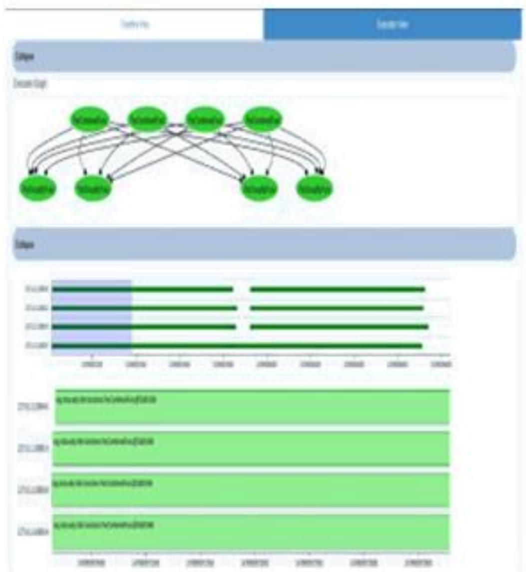

Figure 3: Execution Traces of HDM Applications.

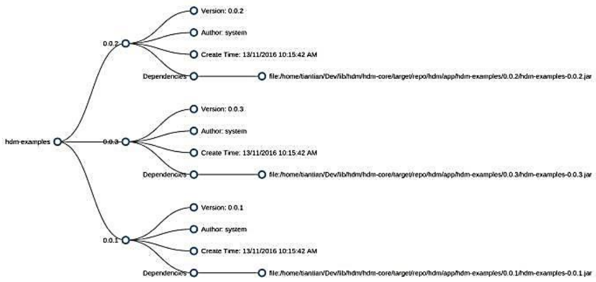

Figure 4: Dependency Management and Version Control of HDM.

for each execution instances of the HDM applications (Fig-ure 2).

Execution History Tracing. In the third part of the demo, we will show how the execution process can be tracked dur-ing and after execution. The HDM server collects and stores the runtime information for each execution task and struc-tures them into DAG based on the task dependencies. Dur-ing or after the execution of the tasks, the HDM server also updates the status in the stored meta-data when it has re-ceived the noti cation messages. The HDM console also summarizes those information and presented it into a view of execution lanes for each core of the workers (Figure 3).

Dependency Management and Version Control. In the last part of the demo, we will show how the HDM server manages the dependencies and provides version control for submitted applications. The dependency and history manager stores all the updating history of each HDM applications and or-ganizes them into a tree based structure. As a result, ad-ministrator users are able to query, analyze and reproduce the historical HDM applications using those dependencies information (Figure 4).

Besides the framework demonstration, we will also dis-cuss in more details about the design choices that we have made on de ning the di erent components of the framework. In addition, performance comparison with the Spark frame-

work [6], using the example scenario, will be presented to demonstrate the e ciency of the HDM optimization tech-niques.

IV. 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.

V. 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.