RESEARCH ARTICLE OPEN ACCESS

Hierarchically Distributed Data Matrix Scheme for Modeling and Building Data Processing Applications

P Chandrashaker Reddy [1], Yadala Sucharitha [2]

[1] Department of CSE, CMR College of Engineering & Technology,

[2] Department of CSE, CMR Institute of Technology, Hyderabad, TS – India.

ABSTRACT

MapReduce is a programming paradigm and an affiliated Design for processing and making substantial data sets. It operates on a large cluster of specialty machines and is extremely scalable Across the past years, MapReduce and Spark have been offered to facilitate the job of generating big data programs and utilization. However, the tasks in these structures are roughly described and packaged as executable jars externally any functionality being presented or represented. This means that extended roles are not natively composable and reusable for consequent improvement. Moreover, it also impedes the capacity for employing optimizations on the data stream of job orders and pipelines. In this article, we offer the Hierarchically Distributed Data Matrix (HDM), which is a practical, strongly-typed data description for writing composable big data appeals. Along with HDM, a runtime composition is presented to verify the performance of HDM applications on dispersed infrastructures. Based on the practical data dependency graph of HDM, various optimizations are employed to develop the appearance of performing HDM jobs. The empirical outcomes show that our optimizations can deliver increases of between 30% to 80% of the Job-Completion-Time for various types of applications when associated with the current state of the art, Apache Spark.

Keywords: - **Map** Reduce, Apache Spark, HDM, Big data processing, Distributed system.

I. 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. For about a decade, the Map Reduce 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 Map Reduce framework is to move analysis to the data, rather than moving the data to a system that can analyze it. It allows programmers data centric fashion transformations data details of distributed execution and fault tolerance by the framework [1]. 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 with new constituted wave of mostly domain-specific, optimized big data processing platforms. In recent years, several frameworks (e.g. Spark, Flink, Pregel, Storm) have been presented to tackle the ever larger datasets on using distributed clusters of commodity machines. However, in reality, many real-world scenarios require pipelining and integration of multiple big data jobs. There are more challenges when applying big data technology in practice [2]. However, in current big data platform such as Map Reduce and Spark, there is no proper way to share and expose a deployed and well-tuned online component to other developers. Therefore, there is massive and even unseen redundant development in big data applications. In addition, as the pipeline evolves, each of the online components might be updated and re-developed, new components can also be added in the pipeline [3]. As a result, it is very hard to track and check the effects during the evolving process. Google's recent report shows the challenges and problems that they have encountered in managing and evolving large scale data analytic applications. Furthermore, as the pipeline become more and more complicated, it is almost impossible to manually optimize the performance for each component not mentioning the whole pipeline. To address the auto optimization problem, Tez and Flume-Java were introduced to optimize the DAG of Map Reduce-based jobs while Spark relies on Catalyst to optimize the execution plan of Spark-SQL. To sum up, the main challenges for current complicated analytic applications can be listed below: Many real-world applications require a chain of operations or even a pipeline of data processing programs. Optimizing a complicated job is difficult and optimizing pipelined ones are even harder [4].

A. Research Challenges

1. Additionally, manual optimizations are time-consuming and error prone and it is almost impossible to manually optimize every program. Integration, composition and interaction with big data programs/jobs are not natively supported: Many practical data analytics and machine learning algorithms require combination of multiple processing components each of which is responsible for a certain analytical functionality.

2. A key limitation for existing frameworks such as Map Reduce and Spark is that jobs are roughly defined and packaged as binary jars and executed as black-boxes without exposing any information about the functionalities. As a result of this, deployed jobs are not natively composable and reusable for subsequent development and integration.
3. Additionally, manual optimizations are time-consuming and error prone and it is almost impossible to manually optimize every program. Integration, composition and interaction with big data programs/jobs are not natively supported: Many practical data analytics and machine learning algorithms require combination of multiple processing components each of which is responsible for a certain analytical functionality.
4. A key limitation for existing frameworks such as Map Reduce and Spark is that jobs are roughly defined and packaged as binary jars and executed as black-boxes without exposing any information about the functionalities. As a result of this, deployed jobs are not natively Composable and reusable for subsequent development and integration.
5. Maintenance and management of evolving big data applications are complex and tedious. In a realistic data analytic process, data scientists need to explore the datasets and tune the algorithms back and force to find out a more optimal solution.

In order to tackle the above challenges, we believe that by improving the basic data and task models, these problems could be addressed to a great extent at the big data execution engine level. In particular, we present the Hierarchically Distributed Data Matrix (HDM) along with the system implementation to support the writing and execution of Composable and integrable big data applications [5, 6]. HDM is a light-weight, functional and strongly-typed meta-data abstraction which contains complete information (such as data format, locations, dependencies and functions between input and output) to support parallel execution of data driven applications. Exploiting the functional nature of HDM enables deployed applications of HDM to be natively integrable and reusable by other programs and applications. The execution graph and functional semantics of multiple optimizations are automatically improving 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 [7, 8].

B. Objective of the Research

The frameworks such as Map Reduce and Spark have been introduced to 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. Deployed jobs are Composable and reusable for subsequent development. The ability optimizations on the data flow of job sequences and pipelines. We present the Hierarchically Distributed Data Matrix which is a functional, strongly typed data representation for writing Composable big data applications. Frameworks provided an execution and 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.

II. BACKGROUND

A. Big data processing frameworks

Several frameworks have been developed to provide processing systems for large distributed records. MapReduce is a large and widely used model for processing records and has been a pioneer in this field. It uses major cost pairs because simple statistics are coordinated during processing. Map and Reduce are primitive and can be inherited from practical programming [9]. In terms of overall performance, Hadoop / Map-Reduce functions are not guaranteed to be fast. All intermediate records during execution are written to a dedicated garage to allow repair of accidents. This is a change that sacrifices efficient use of memories and the neighborhood garage. The Map-Reduce window is usually not effective for instant and small functions where you may remember the facts [10]. Spark uses memories because important data is stored during execution. Therefore, it can offer much higher performance, compared to the functions running in MapReduce. Spark's basic programming abstraction is called Flexible Distributed Data Sets (RDDs), which are a logical grouping for dividing statistics between machines. In addition, programs in Spark are divided as DAGs mainly on Stage which can be separated with the help of shuffle dependencies. In the activity rationalization process, Spark also combines parallel processes into a single task, where the experiment also achieves the same improved feature integration as it did during HDM. However, similarly, record chain improvements are not provided along with reordering and rewriting within the Spark processing engine [11].

B. Attributes of HDM

Basically, HDM is represented as HDM [T, R], where T and R, are two facts to enter and exit, respectively. The HDM itself represents a function that transforms statistics Input to the output in addition to those basic features, HDM addition Includes information such as statistical units, neighborhood, and distribution to guide improvement and implementation. HDM supports the following simple functions: Functional: HDM is largely an illustration based of the function that calculates the output Some HDM account input is approx. Evaluate the property contained in the income dataset. While calculating HDM, Appearance results are not attributed. Strongly written: HDM consists of at least clear facts Types and input type and output type, which can be drawn of input and output formats based mainly closed function. They are necessary to ensure compatibility of types of information [12]. Portable: HDM is a neutral component containing Complete Facts for a Computer Project. And so on, in principle, HDMis removable and can be moved to any node within the context of the executable HDM. Site aware: HDMs consist of facts Proximity (represented by rich URL) of entries and exits. Although some information about the place is better available during runtime, it facilitates application optimization for simultaneous information sites around Develop level plans [13].

C. Optimizations for Big Data Applications

Flume-Java It's a Java library that Google recently delivered. Designed in Map-Reduce, it offers better stage setup and a host of improvements to higher implementation plans. The overall performance advantages of these enhanced plans are manually enhanced Map-Reduce functions, but Flume-Java has freed programmers from the refined optimization method and is often boring. Tez is a graphics-based optimizer that can significantly improve Map-Reduce's work written in Pig and Hive [14]. Basically, Tez simplifies Map-Reduce tubes by combining a pair of specific distillation devices and redundant reducers. It also instantly links the results of previous work to subsequent work, which reduces the rate of writing the metadata in HDFS, and thus, can improve the overall performance of the work performed. Apache MRQL6 is a framework delivered as a framework for improving query performance and processing for extensive and distributed information analysis, and is based on Apache Hadoop, Spark, Hama and Flink [15]. Specifically, it offers a query language similar to SQL that can be devalued in neutral modes: Map-Reduce mode using Apache Hadoop, Spark mode with Apache Spark, BSP mode, Use Apache Hama and Flink mode with Apache Flink. The rest of the article is organized as follows. The third section describes related work and the subsequent section explains the proposed methodology adopted. The fifth section outlines the experimental outcomes and discussions while, the conclusion is given in the last section.

III. RELATED WORKS

Apache Tez has been fortunate to learn from the development and experiences of similar systems such as Dryad, Hyracks and Nephele. All of them share the concept of modeling data processing as DAGs with vertices representing application logic and edges or channels representing data transfer [16]. Tez makes this more fine-grained by adding the concepts of inputs, processor and outputs to formally define the tasks executing the DAGs, leading to clear separation of concerns and allowing pluggable task composition. All of them participate to varied extents in the data plane and define some form of data format, which allows applications to define custom formats that derive from the base definition. All of them define on-disk, over-network and in-memory communication channels [17]. Tez, on the other hand, does not define any data format and is not part of the data plane at all. On a similar note, Hyracks defines an operator model for execution that allows it to understand the data flow better for scheduling. Tez treats processors as black boxes so that the application logic can be completely decoupled from the framework. Nephele is optimized for cloud environments where it can elastically increase or decrease resources and choose appropriate virtual machines. Tez also enables resource elasticity by acquiring and releasing resources in YARN [18].

Dryad and Tez share the concept of vertex managers for dynamic graph re-configurations. Tez takes this concept a step further by formalizing an API that allows the managers to be written without knowing the internals of the framework and also defining an event based communication mechanism that enables application code in tasks to communicate with application code in vertex managers in order to actuate the re-configurations [19]. In addition, Tez adds the concept of input initializers to formally model primary data sources and apply runtime optimizations while reading them. Dryad schedules tasks when all the inputs of the tasks are ready to be consumed, to prevent scheduling deadlocks. Tez allows out of order execution for performance reasons and has built-in preemption to resolve scheduling deadlocks. Overall, Tez differs from these systems in its modeling capabilities and the design goal of being a library to build engines rather than being an engine by itself. MapReduce is, of course, the incumbent engine in the Hadoop ecosystem [20].

Tez subsumes the MapReduce APIs such that it is possible to write a fully functional MapReduce application using Tez. Dremel [21] is a processing framework for interactive analysis of large data sets based on multi-level execution trees that is optimized for aggregation queries and has motivated systems like Presto and Apache Drill [22]. These, and other SQL query engines like Impala or Apache Tajo [23], differ from Tez by being engines optimized for specific processing domains whereas Tez is a library to build data processing applications. Spark is a new general purpose data processing engine. It exposes a Resilient Distributed Dataset (RDD) based computation model that eventually gets executed on an in-memory storage and compute engine [24]. Tez, again differs being a library and not a general purpose engine. Tez does not provide any storage service but applications can use existing inmemory stores, e.g. HDFS memory storage, to get the advantage of in-memory computing. The Spark notion of using RDDs as a means of implicitly capturing lineage dependency between steps of processing can be related to capturing that same dependency explicitly via defining the DAG using Tez APIs. An important category of systems to compare against are other frameworks to build YARN-applications [25]. The two most relevant in this space are Apache REEF [19] and Apache Twill [31]. These systems focus on a much broader class of applications (including services), than Tez, and thus provide a lower-level API. Tez focuses on supporting data-flow driven applications, and thus consciously chooses to provide a structured DAG-based control-flow [26].

IV. METHODOLOGY

We present the Hierarchically Distributed Data Matrix along with the system implementation to support the writing and execution of compostable and integrable big data applications. HDM is a light-weight, functional and strongly-typed meta-data abstraction which contains complete information to support parallel execution of data driven applications. 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 [27].

In particular, the main contributions of this paper can be summarized as follows: HDM, a light, practical and strongly written statistical representation of growth and description of parallel information packets. A runtime framework to support the implementation and integration of HDM applications in distributed environments. A set of enhancements that rely mainly on a dependency chart of useful information to improve the performance of HDM functions. An empirical evaluation of the performance of HDM compared to the contemporary field of artwork for large information processing structures, Apache Spark. HDM programs are designed to be interactive at some point Runtime asynchronously. Specifically, HDM Applications can be written and integrated into other programs as a normal code chipset. Then, by activating the motion. Interfaces, functions are sent dynamically to the relevant implementation context that can be either a group of multicenter yarns or a group of workers.

A. Categorization of HDM

In principle, HDM is an entire tree-based structure consisting of node forms: Distributed Data Matrix: Paper nodes in the HDM hierarchy preserve real information and are responsible for the appearance of atomic processes in information blocks. Distributed Functional Matrix (DFM): Each non-paper node maintains the performance and distribution of relationships for HDMs for children; during implementation, it is also responsible for collecting and adding secondary node results as necessary. From a useful point of view, DDM is seen as a feature that maps a path to a real data set. Basically, DDM can be represented as HDM [Route, T]. During implementation, log parsers are encapsulated to load information from the address information according to their protocols and then rework the entry to the expected outgoing DDM formats. DFM is considered as a high-level representation that focuses on the deliberate reliance of human development mechanisms on the planning stages. Before implementation, DFM can be similarly explained as DDM dependencies according to realities and expected parallelism. The separation between DFM and DDM provides different degrees of views to help in the exclusive ranges of planning and optimization. In addition, the hierarchy of DFM and DDM also ensures that a close account in the truth node does not care about the transfer of information and coordination between siblings, thus, leaving the privileged nodes free to use the assembly steps are shown in Figure 1.

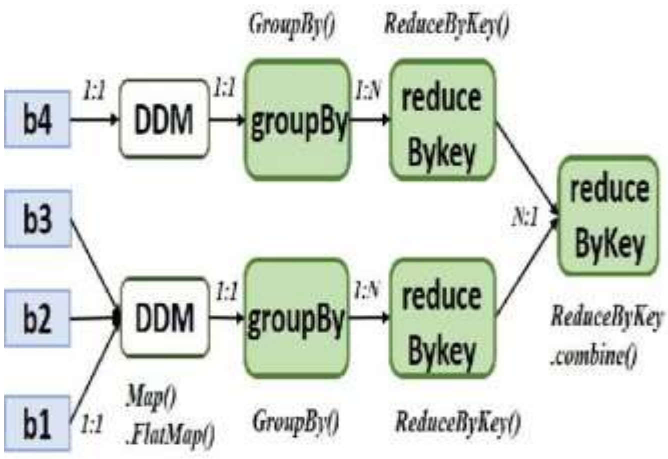

Fig. 1. Word-count after function fusion Data flow

In HDM, half of the parallel operations as a series of Operations that start with One-To-One or N-To-One statistics Count and give up the facts one by one or one to N Accreditation. These parallel processes can be integrated into HDM instead of being in a separate HDM. Parallel Capabilities that include map, search, filter, neighborhood drop / group It will be linked directly to the main nodes until it reaches Root or trip through the dependency N-to-N and One-To-N. The Parallel merge base in HDM can be as accurate as: Parallel Fusion Rule: Assumed two associated HDM $HDM1[T;\ R]$ with function $f\colon T \to R$ tracked by $HDM2[R;U]$ with function $g\colon R \to U$ If the dependency Between them they are one by one, then can be combined $HDMc[T;U]$ with function $f\ g : T \to U$. This rule can be applied repeatedly in a series of Parallel processes to obtain the final combined HDM [28].

B. Runtime Engine

Basic commitment to additives at run time the engine is coordination and collaboration between tasks so that Unique functions like HDM can be effectively complete. Fig.2 Runtime Executing process of HDM Jobs Fig.2, demonstrates the main process of implementing HDM functions within its Machine runtime as described, the main levels of implementation HDM works consist of plans for logical creation, improvement, and physics Develop plans, programming and implementation. Before implementation, HDMs must be defined as executable the responsibilities of executives. The rationalization system is particularly divided into two sub-steps: logical and construction plans Body layout is shown in Figure 2.

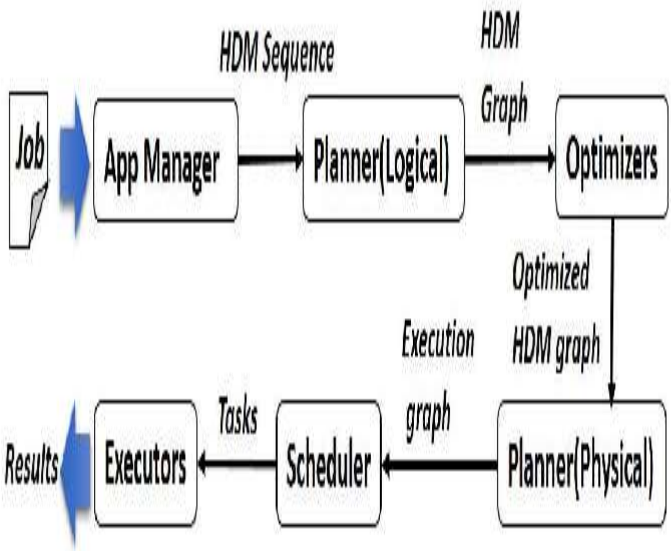

Fig. 2. Runtime Executing process of HDM Jobs

V. EXPERIMENTAL RESULTS

Experiments are conducted on vast scale and results are displayed in Figure 3, 4, 5 and 6.

Fig. 3. Registration form

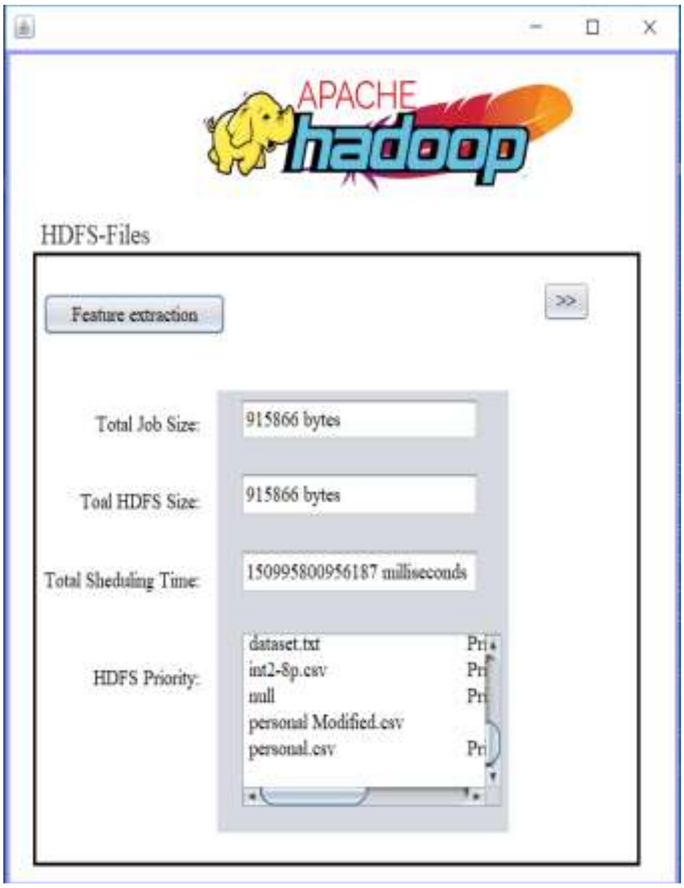

Fig. 4. HDFS files design output

Fig. 5. HDM will take less time to perform when compared to Map reducer

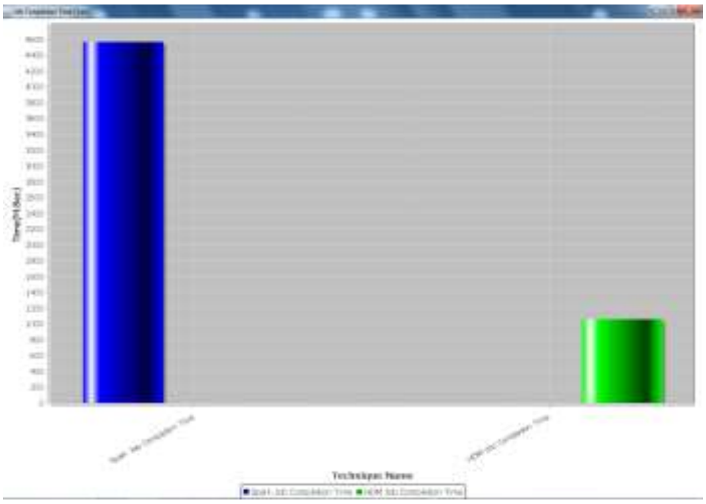

Fig. 6. Job completion chart

VI. CONCLUSION

In this paper, we have performed HDM as a practical and strongly-typed data description, simultaneously with a runtime system implementation to maintain the execution, optimization, and administration of HDM applications. Based on the scientific nature, apps signed in HDM are natively composable and can be combined with current applications. At the same time, HDM functionality streams are automatically optimized before they can be achieved in the runtime system. In addition, HDM programming frees builders from the tedious allocation of integration and improves information-based application guides so that they can focus on good software governance and log analysis algorithms. Finally, the overall performance evaluation refers to the sharp performance of HDM in the Spark evaluation specifically for targeted operations that contain groups and filters. However, HDM is still at its initial level of improvement, as some limitations must be resolved in our target work: 1) disk-based processing wants to be supported if the group's general memory is not retrieved for terribly large functions; 2) fault tolerance should be considered a condition Critical for practical use; 3) The long-term challenge we plan to solve is to develop improvements to address units of ad hoc records in a heterogeneous manner, which usually cause heavy outliers and seriously slow down the time of glory crowning action in general and smashing the use of global aid.

REFERENCES

1. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, JohannChristoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinlander, Matthias J. Sax, Sebastian Schelter, Mareike ¨ Hoger, Kostas Tzoumas, and Daniel Warneke. The Stratosphere ¨ platform for big data analytics. VLDB J., 23(6), 2014.
2. Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational Data Processing in Spark. In SIGMOD, pages 1383– 1394, 2015.
3. Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. In PLDI, 2010.
4. Jeffrey Dean and Sanjay Ghemawat. Map Reduce: simplified data processing on large clusters. Commun. ACM, 51(1), 2008.
5. Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, ¨ Eric N. Hanson, Owen O'Malley, Jitendra Pandey, Yuan, Rubao Lee, and Xiaodong Zhang. Major technical advancements in Apache Hive. In SIGMOD, pages 1235–1246, 2014.
6. Mohammad Islam, Angelo K. Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur. Oozie: towards a scalable workflow management system for Hadoop. In SIGMOD Workshops, 2012.
7. Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD Conference, 2010.
8. Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In SIGMOD, 2008.
9. Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun C. Murthy, and Carlo Curino. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In SIGMOD, 2015.
10. Sherif Sakr and Mohamed Medhat Gaber, editors. Large Scale and Big Data - Processing and Management. Auerbach Publications, 201
11. Dongyao, Liming Zhu, 2015, "Composable and Efficient Functional Big Data Processing Framework".
12. Matei Zaharia, Michael J. Franklin, 2010, "Spark: Cluster Computing with Working Sets", pp.1-7.
13. Bikas Saha, Arun C. Murthy, and Carlo Curino, 2015, "Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications",pp. 1357-1369.
14. Reddy PC, Sureshbabu A. An applied time series forecasting model for yield prediction of agricultural crop. InInternational Conference on Soft Computing

and Signal Processing 2019 Jun 21 (pp. 177-187). Springer, Singapore.

15. Alexander Alexandrov, Odej Kao, Matthias J, 2014," The Stratosphere platform for big data analytics" VLDB J.,23(6).
16. Reddy PC, Babu AS. Survey on weather prediction using big data analystics. In2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) 2017 Feb 22 (pp. 1-6). IEEE.
17. Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD Conference, 2010.
18. Reddy PC, Sureshbabu A. An applied time series forecasting model for yield prediction of agricultural crop. InInternational Conference on Soft Computing and Signal Processing 2019 Jun 21 (pp. 177-187). Springer, Singapore.
19. Sherif Sakr, Anna Liu, and Ayman G. Fayoumi. The family of map reduce and large-scale data processing systems. ACM CSUR, 46(1):11, 2013.
20. Reddy PC, Sureshbabu A. An Adaptive Model for Forecasting Seasonal Rainfall Using Predictive Analytics. InInternational Journal of Intelligent Engineering and Systems 2019 (pp. 22-32).
21. Grzegorz Malewicz, Naty Leiser, and Grzegorz Czajkowski, 2010, "Pregel: a system for large-scale graph processing", pp.135-145.
22. Reddy PC, Babu AS. A novel approach to analysis district level long scale seasonal forecasting of monsoon rainfall in Andhra Pradesh and Telangana. International Journal of Advanced Research in Computer Science. 2017 Nov 1;8(9).
23. Matei Zaharia, Murphy McCauly, and Ion Stoica, 2012, Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing".
24. M. Zaharia, J. Sen Sarma, and I. Stoica, 2010, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling."
25. Sucharitha Y, Vijayalata Y, Prasad VK. Analysis of Early Detection of Emerging Patterns from Social Media Networks: A Data Mining Techniques Perspective. InSoft Computing and Signal Processing 2019 (pp. 15-25). Springer, Singapore.
26. D. G. Murray, M. Schwarzkopf, S. Hand, 2011," Ciel: a universal execution engine for distributed data-flow computing".
27. Sucharitha Y, Prasad VK, Vijayalatha Y. Emergent Events Identification in Micro-Blogging Networks Using Location Sensitivity.
28. Norin S, inventor; Microsoft Corp, assignee. System and method for the distribution of hierarchically structured data. United States patent US 5,812,773. 1998 Sep 22.