*Research Paper*
*Impact Factor: 3.567*
*Peer Reviewed Journal*

*IJMDRR*
*E- ISSN –2395-1885*
*ISSN -2395-1877*

# A NOVEL TECHNIQUE TO PERFORMING BIG DATA APPLICATIONS BASED ON MAP-REDUCE FRAMEWORK AND WEIGHTED ROUND ROBIN

**J. Savitha*** **G. Mahalakshmi****

**Assistant Professor, Department of Computer Science, Kovai Kalaimagal College of Arts & Science, Coimbatore.*
***M Phil Scholar, Department of Computer Science, Kovai Kalaimagal College of Arts & Science, Coimbatore.*

***Abstract***

*A big data generally known as to perform into the large scale distributed systems and processing the large amount of data. Google's MapReduce and Apache's Hadoop, its open-source implementation, are the defector software systems for big-data applications. The map-reduce framework frequently generates large number of intermediate data's. Normally, we can combine the process of big data applications into map reduce for accessing more number of processes at the time. This map reduce framework we can use two types of processes. There are map cache and also reduce cache. These processes are combined to solve the smaller applications. To increase the speed up process and also avoid the generation of redundant numbers we have to move onto the proposed system.*

***Key Words: Memory Reduce, Less Memory, Less Process Time, Reduce Framework.***

## INTRODUCTION

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Big Data refers to the massive amounts of data collected over time that are difficult to analyze and handle using common database management tools. The data are analyzed for marketing trends in business as well as in the fields of manufacturing, medicine and science. The types of data include business transactions, e-mail messages, photos, surveillance videos, activity logs and unstructured text from blogs and social media, as well as the huge amounts of data that can be collected from sensors of all varieties. The storage industry is continuously challenged as Big Data increases exponentially. While the physical storage can be enhanced with more terabyte drive arrays, the software infrastructure must be flexible enough to quickly and economically accommodate ever greater volumes of transactions and queries. The analytical challenge is deriving meaningful information from data in petabyte and exabyte volumes. Big Data analytics breaks down the data sets into smaller chunks for efficient processing and employs parallel computing to derive intelligence for effective decision-making.

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate.

Values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines.

The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

Storage –One primary component of the Hadoop ecosystem is HDFS—the Hadoop Distributed File System. The HDFS allows users to have a single addressable namespace, spread across many hundreds or thousands of servers, creating a single large file system. HDFS manages the replication of the data on this file system to ensure hardware failures do not lead to data

loss. Many users will use this scalable file system as a place to store large amounts of data that is then accessed within jobs run in Hadoop or by external systems. Database –The Hadoop ecosystem contains components that allow the data within the HDFS to be presented in a SQL-like interface. This allows standard tools to INSERT, SELECT, and UPDATE data within the Hadoop environment, with minimal code changes to existing applications. Users will commonly employ this method for presenting data in a SQL format for easy integration with existing systems and streamlined access by users.

The next lower level of the memory hierarchy is the main memory which is large but also comparatively slow. While external memory such as hard disk drives or remote memory components in a distributed computing environment represent the lower end of any common hierarchical memory design, this paper focuses on optimization techniques for enhancing cache performance. The levels of the memory hierarchy usually subset one another so that data residing within a smaller memory are also stored within the larger memories. EÆcient program execution can only be expected if the codes respect the underlying hierarchical memory design. Unfortunately, today's compilers cannot introduce highly sophisticated cache {based transformations and, consequently, much of this optimization deport is left to the programmer.

**Literature Survey**

**1. J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. of ACM, vol. 51, no. 1, pp. 107-113, 2008.**

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

**2. H. Gonzalez, A. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen, Google fusion tables: Data management, integration and collaboration in the cloud, in Proc. of SOCC'2010, New York, NY, USA, 2010.**

Google Fusion Tables is a cloud-based service for data management and integration. Fusion Tables enables users to upload tabular data files (spreadsheets, CSV, KML), currently of up to 100MB. The system provides several ways of visualizing the data (e.g., charts, maps, and timelines) and the ability to filter and aggregate the data. It supports the integration of data from multiple sources by performing joins across tables that may belong to different users. Users can keep the data private, share it with a select set of collaborators, or make it public and thus crawlable by search engines. The discussion feature of Fusion Tables allows collaborators to conduct detailed discussions of the data at the level of tables and individual rows, columns, and cells. This paper describes the inner workings of Fusion Tables, including the storage of data in the system and the tight integration with the Google Maps infrastructure.

The main aspects of the architecture of Fusion Tables, a cloud-based data management service that focuses on the collaborative aspects of data management. Both the challenges we face and the advantages we gain stem from the fact that we do our best to integrate Fusion Tables with the existing Google infrastructure. Using Big table and Megastore provides us with a scalable and replicated data store (albeit not for very high transaction rates). On the other hand, this makes it trickier to implement certain kinds of SQL queries. The Google Maps infrastructure enables us to provide a user experience that integrates seamlessly with other map features. Moving forward, our goal is to provide a data management experience that seamlessly integrates with other experiences on the Web. We believe that integration with the Web is a promising approach to making data management applicable to a broad set of users. In addition to integration with Google Maps, this means integration with search (e.g., our public tables are already crawlable) and with enterprise collaboration tools such as Google Docs.

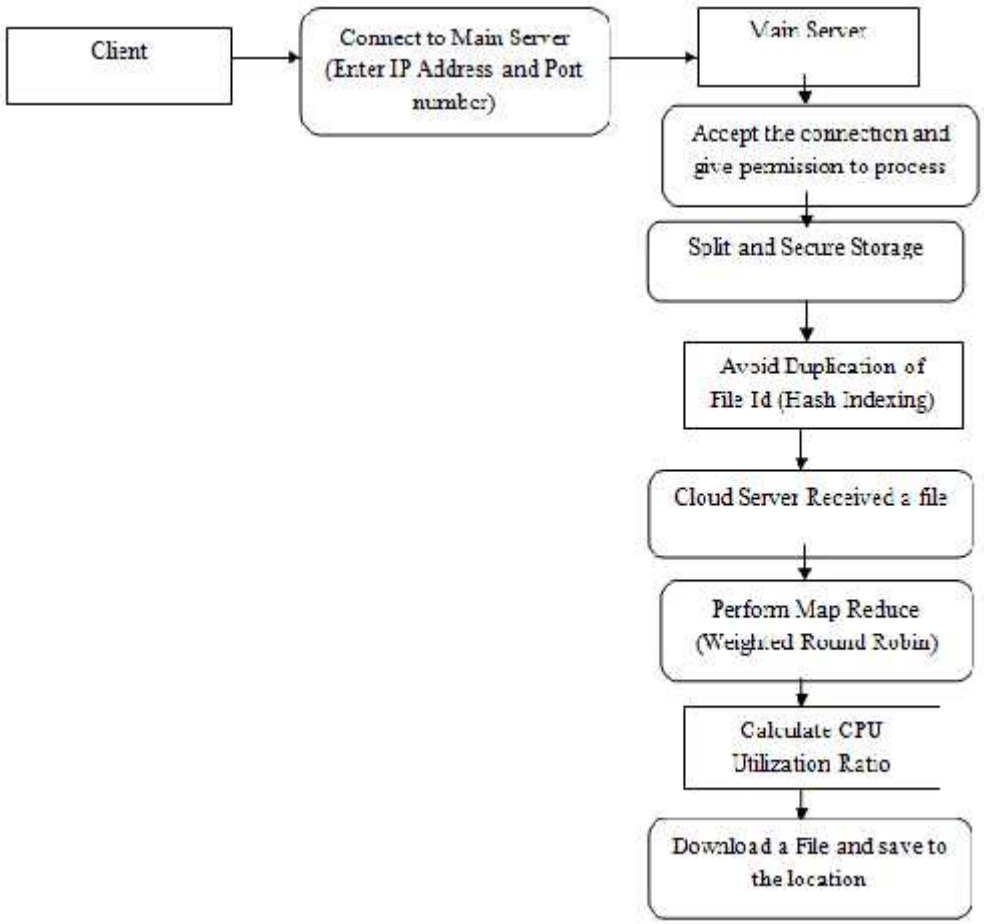
**3. D. Battr'e, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, Nephele/pacts: A programming model and execution framework for web-scale analytical processing, in Proc. of SOCC'2010, New York, NY, USA, 2010.**
We present a parallel data processor centered around a programming model of so called *Parallelization Contracts* (PACTs) and the scalable parallel execution engine Nephele. The PACT programming model is a generalization of the well-known map/reduce programming model, extending it with further *second-order functions*, as well as with *Output Contracts* that give guarantees about the behavior of a function. We describe methods to transform a PACT program into a data flow for Nephele, which executes its sequential building blocks in parallel and deals with communication, synchronization and fault tolerance. Our definition of PACTs allows to apply several types of optimizations on the data flow during the transformation. The system as a whole is designed to be as generic as (and compatible to) map/reduce systems, while overcoming several of their major weaknesses: 1) The functions map and reduce alone are not sufficient to express many data processing tasks both naturally and efficiently. 2) Map/reduce ties a program to a single fixed execution strategy, which is robust but highly suboptimal for many tasks. 3) Map/reduce makes no assumptions about the behavior of the functions. Hence, it offers only very limited optimization opportunities. With a set of examples and experiments, we illustrate how our system is able to naturally represent and efficiently execute several tasks that do not fit the map/reduce model well.

**Algorithm**

**Weighted Round Robin Algorithm**

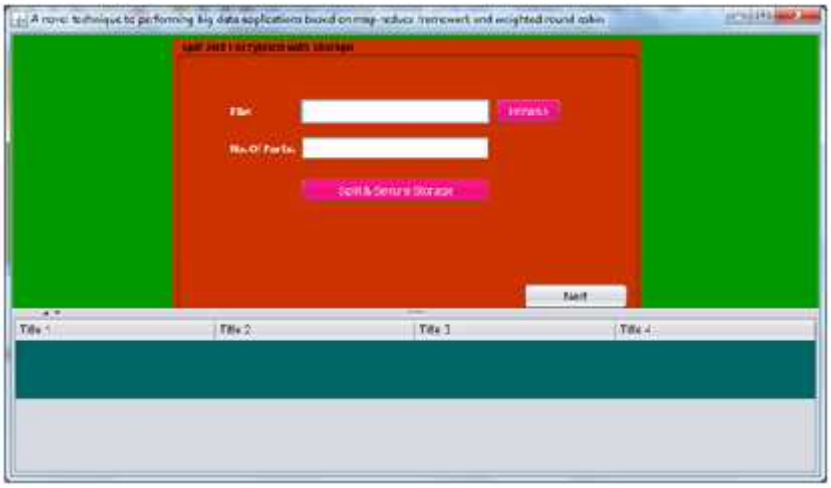
The WRR Scheduling algorithm uses the advantage of RR in eliminating the problem of starvation. It then implements a priority-based technique in several areas. All processes in the job queue are given time on the processor in the form of a time slice, thereby eliminating concern for starvation. Also short processes that get re-queued often will not have to wait for longer higher priority processes since all jobs in the queue will be granted time on the processor.

**Data Flow Diagram**

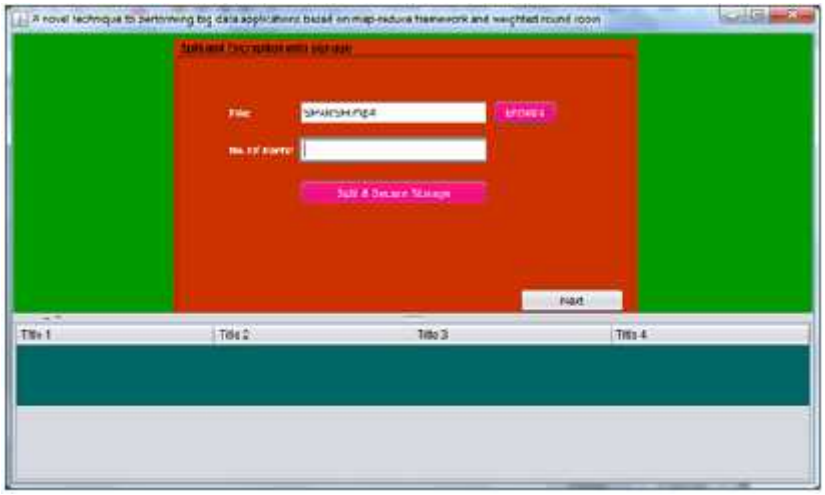


IJMDRR
E- ISSN –2395-1885
ISSN -2395-1877

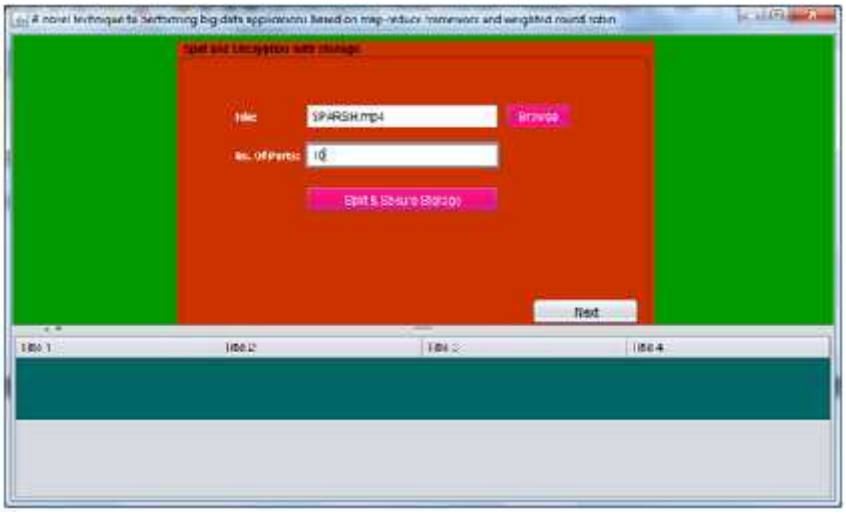
**Experimental Output**
**Run Storage.java**



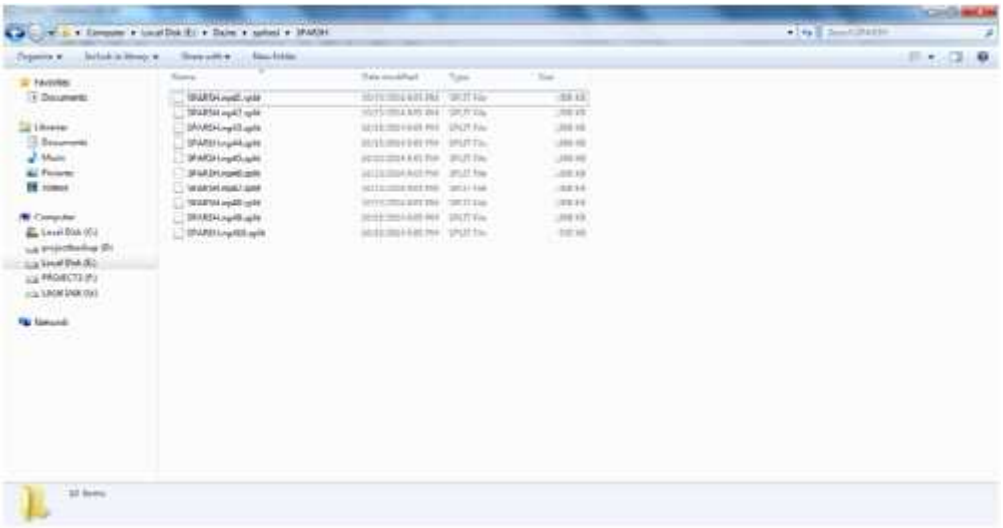
**Get Input**

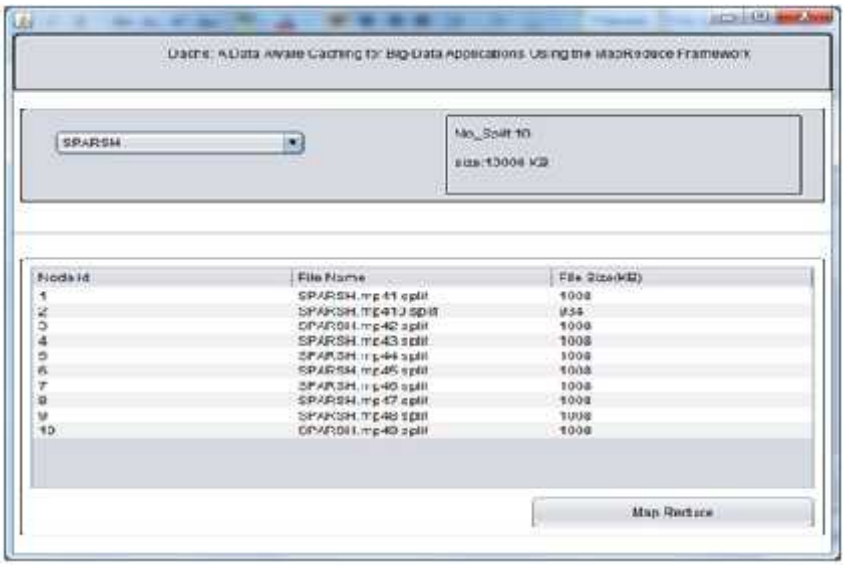


**Give no of spilted file is 10**

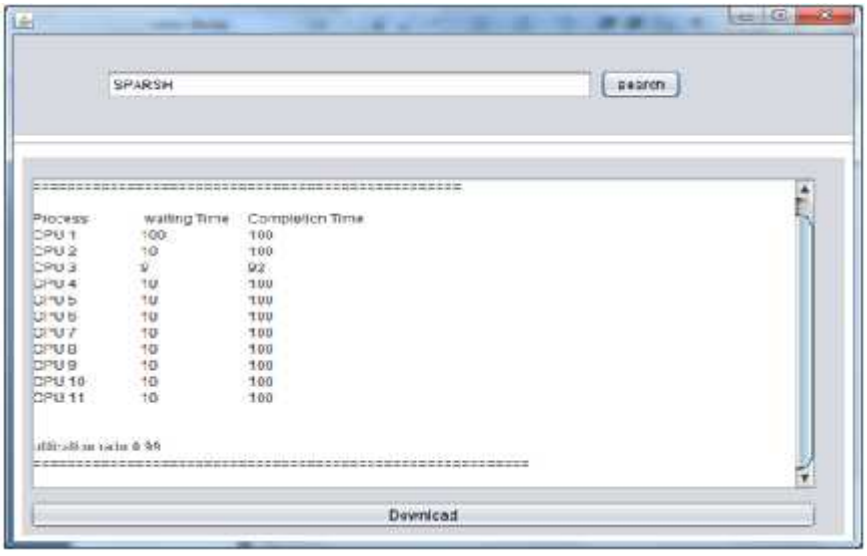


IJMDRR
E- ISSN –2395-1885
ISSN -2395-1877






**Output**



**Fesibility Study**

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault.

**References**

1. J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi‘eres, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, The case for ramcloud, Commun. of ACM, vol. 54, no. 7, pp. 121-130, 2011.
2. L. Popa, M. Budiu, Y. Yu, and M. Isard, Dryadinc: Reusing work in large-scale computations, in Proc. Of HotCloud’09, Berkeley, CA, USA, 2009.
3. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Trans. Computer Systems, 26(2), 2008.
4. J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. Foundations and Trends in Databases, 1(4):379{474, 2009.
5. T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In Proc. NSDI, 2010.