Synopsis for the title

# "Performance Optimization of Big Data Processing using Heterogeneous Distributed Computing"

Submitted by

Mr. Ankit Shah [FOTE/916]

as a partial fulfillment for

PhD in Computer Science and Engineering

at

The Maharaja Sayajirao University of Baroda

Research Guide

Dr. Mamta Chandraprakash Padole

Associate Professor

Department of Computer Science and Engineering

Faculty of Technology and Engineering

The Maharaja Sayajirao University of Baroda

## List of Figures

## List of Tables

# Table of Contents

# Chapter 1: Introduction

## 1.1 Distributed Computing

Distributed Computing (DC) refers to computation using system of loosely coupled computers striving to solve computationally intensive problems that are difficult to be computed using single computer. Distributed computing is used to solve complex computational problems that cannot be solved within a specified time frame on a single computer. The complex computational problems may involve either compute intensive or data intensive processing.

### 1.1.1 Distributed Computing System (DCS)

“A Distributed Computing System, also referred as a Distributed System, is a collection of independent computers that appears to its end users, as a single computing system” [1].Distributed Computing System (DCS) refers to a system of multiple computers working on a single problem that is computationally intensive. Distributed Computing System is a wide scale infrastructure that supports sharing of resources, distribution transparency, scalability, single point failure handling and single system image concept in large-scale problem solving. Distributed Computing System provides these advantages compared to traditional centralized computing system.

### 1.1.2 Distributed Computing System Architecture

In computer architecture terminology, distributed computing system belongs to the class of loosely coupled Multiple Instruction, Multiple Data (MIMD) machines, with each node having an unshared memory [2]. Below fig.1 shows a simple architecture of a distributed computing system [1, 3]

Distributed Computing systems are built up using existing hardware, operating systems (OS) and network. These hardware, OS and networks may be of same type or different type i.e. they may be either homogeneous or heterogeneous respectively, in nature. A distributed system comprises of collection of autonomous computers, linked through a computer network and distribution middleware. The middleware enables distribution transparency, where-in the task submitted to master, is distributed amongst multiple slaves. Thus, middleware is the bridge that connects distributed applications across multiple systems. Middleware provides standard services such as naming, persistence, concurrency control to ensure that accurate and faster results for processes are produced.

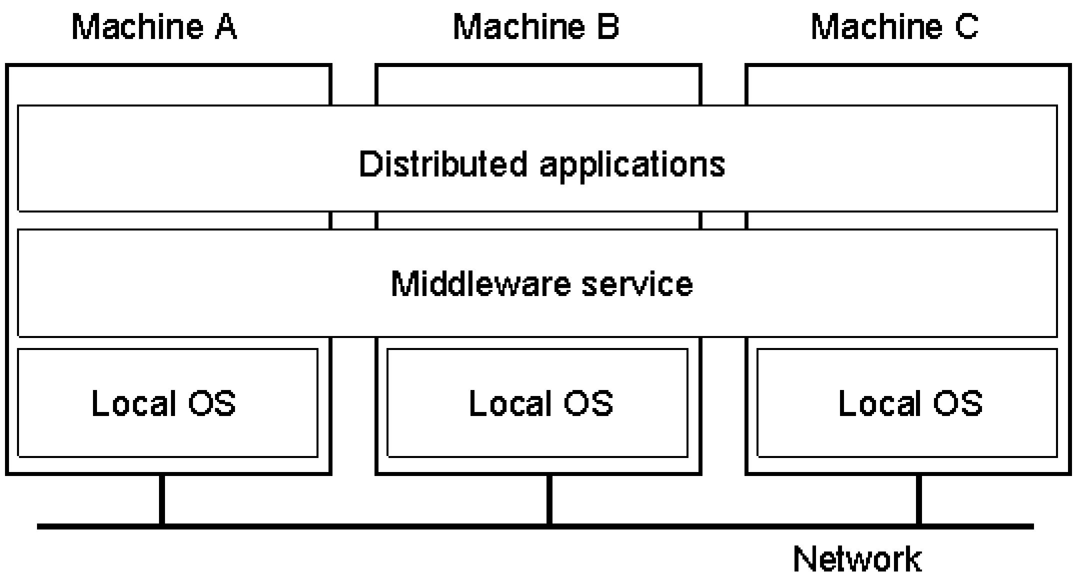


*Figure 1Distributed Computing System Architecture*

Distributed computing system comprises of variety of hardware and software, to form a distributed platform. At a lower level, it is necessary to have multiple CPUs which are interconnected with each other by network. At a higher level, those interconnected CPUs will communicate with each other through Middleware. Distributed computing system can be categorized as homogeneous or heterogeneous based on their hardware, OS, connection, architecture, and other components.

### 1.1.3 Homogeneous Distributed Computing System (HDCS)

A distributed computing system is said to be Homogeneous Distributed Computing System: a) If hardware on each computing machine has same architecture, processing capacity and same storage representation. b) If software (i.e. Operating system, Compiler etc.) on each computing machine has same storage organization and similar operation speed. The requirements for a homogeneous distributed computing system are quite stringent and are frequently not met in network of workstations, or PCs, even when each computer in the network is of the same make and model.

### 1.1.4 Heterogeneous Distributed Computing System (HeDCS)

Heterogeneous Distributed Computing System (HeDCS) is one which is not homogeneous. Heterogeneous platform used for distributed computing always include processors and communication network interconnecting the processors, of different types. Distributed computing systems are naturally heterogeneous. A heterogeneous distributed system is a dedicated system designed mainly for high - performance distributed computing. In this research study the aim is to use Heterogeneous Distributed Computing System (HeDCS), hence it is discussed in detail.

Computing performed using Homogeneous Distributed Computing System (HDCS) is referred as Homogeneous Distributed Computing (HDC) and likewise, the computing performed using Heterogeneous Distributed Computing System (HeDCS) is known as Heterogeneous Distributed Computing (HeDC). Henceforth, for the discussion, the term HeDC will be used to describe problem solving on using HeDCS.

## 1.2 Heterogeneous Distributed Computing

In Practical scenario, it is difficult to find a computing platform, with all computers involved in processing, to be of perfectly uniform configuration. Hence, the emphasis is using heterogeneous set of computational resources, for solving computational intensive problems.

A Heterogeneous Distributed Computing (HeDC) that consists of a heterogeneous suite of processors, high-speed interconnections, interfaces, operating systems, communication protocols and programming environments provides a variety of architectural capabilities, which can be coordinated to  process an application that has diverse execution requirements [4,5]. HeDC is now well recognized as an important computing paradigm, in meeting the computational requirements of many applications in science, engineering and commerce. The examples of applications are weather forecasting, simulation modeling, mapping of the human genome, big data processing, image processing, modeling of semiconductors, superconductors and banking systems [6-9]. While the distributed computing systems offer the promise of vastly increased performance, it introduces additional complexities such as scheduling of parallel program, load balancing among the involved processors, process synchronization, communication, handling data redundancy etc. which are not encountered with stand-alone processing.

A heterogeneous distributed computing system, as shown in fig.2, is a dedicated system designed mainly for high-performance computing, which is obtained from the classical homogeneous system architecture by relaxing one of its three key properties, leading to the situation wherein :

- Processors in distributed environment may not be identical.
- The communication network may have a regular but heterogeneous structure.
- The HeDC may be a multitasking computer system, allowing several independent users to simultaneously run variety of applications, on the same set of processors.

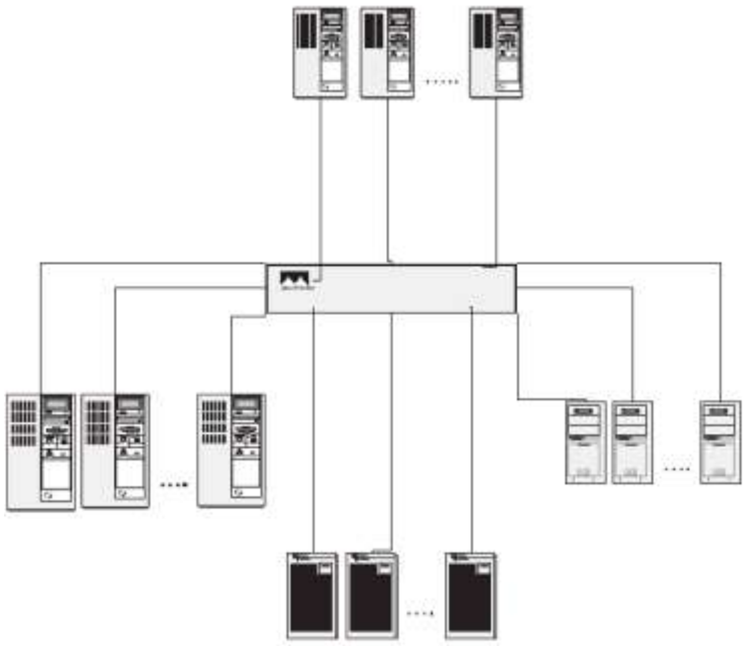


*Figure 2A heterogeneous system with processors of different architectures*

***HeDC Challenges***

Heterogeneous Distributed Computing comes with new challenges due to non-uniformity, variety of programming models, and overall varied system capability. The following factors are to be considered, while applying HeDC for large data sets.

- Different instruction set and memory set architectures
- Library and OS services are not uniformly available on distributed nodes
- CPUs have different performance level and power consumption
- Compute elements have different cache structure, network architecture

Above factors may result in performance degradation while working on large data sets, also referred as Big Data. Big Data processing would be difficult to be performed on a single computer; it also cannot apply upon homogeneous distributed computing, because of scalability issues. Hence, there is a need of HeDC for processing large data sets i.e. Big Data.

## 1.3 Big Data

Big Data is data that exceeds the processing capacity of single system or conventional group of computers. The data is too big, moves too fast and may comprise of unstructured data which may not fit into traditional database and storage structure. In other words, Big Data is an all-encompassing term for any collection of data sets so large and complex that it becomes difficult to process using on-hand data management tools or traditional data processing environments. Big Data solutions are useful for business analytics.

According to an IBM study, nearly 2.5quintillion bytes of data is created every year, so much that about 90% of the data in the world today has been created in the last two years itself [10]. This data comes from everywhere: social networking sites, GPS, sensors, private or public networks etc. Although, Big Data may be in both the forms: structured as well as unstructured, these data are generally in raw form, i.e. they are unstructured data sets. Big data usually include data sets with sizes such as Petabytes or Zetabytes, which are beyond the ability of commonly used software tools to capture, create, manage, and process the data within a tolerable elapsed time.Big data requires cost-effective and innovative forms of information processing.

### 1.3.1 Characteristics of Big Data

Big Data has been defined by the four “V”s [10]: Volume, Velocity, Variety, and Veracity, as shown in fig 3. These four characteristics help to determine whether your information architecture needs to process Big Data.

**Volume:** The amount of data. While volume indicates more data, it is the granular nature of the data that is unique. Big Data requires processing high volumes of low-density data, that is, data of unknown value, such as social networks, clicks on a web page, network traffic, sensor-enabled equipment etc.

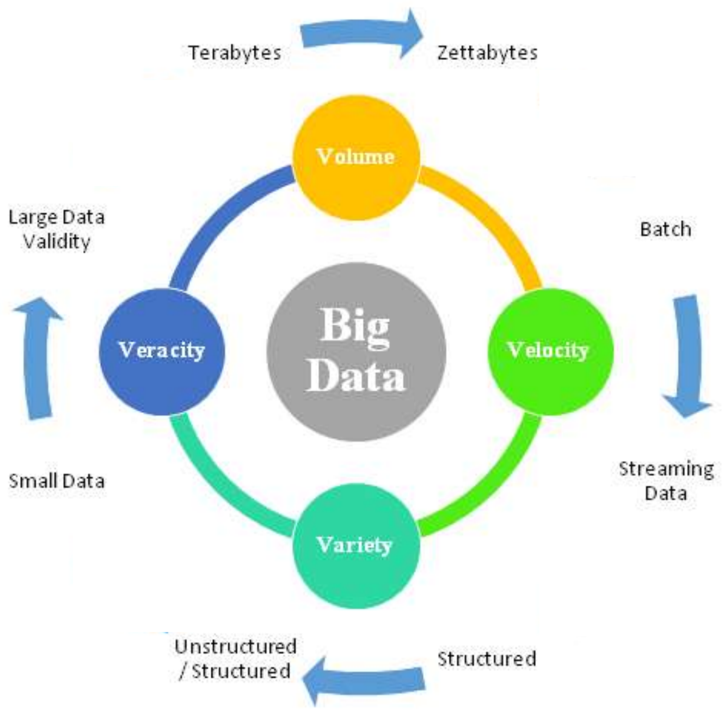


*Figure 3 Characteristics of Big Data*

**Velocity:** A fast rate that data is received and perhaps acted upon. The highest velocity data normally streams directly into memory versus being written to disk.

**Variety:** New unstructured data types. Unstructured and semi-structured data types, such as text, audio, and video require additional processing to both derive meaning and the supporting metadata.

**Veracity:** Veracity refers to the biases, noise and abnormality in data. It is important to check whether the data that is being stored and processed is meaningful to the problem being analyzed. It checks the authenticity of data. Big data solutions must validate the correctness of the large amount of rapidly arriving data.

### 1.3.2 Challenges in Big Data Processing

When data is in large amount (Big Data), it also comes with huge challenges like: data acquisition, storage, management and analysis. Traditional data management systems are based on the relational database management system (RDBMS). However, such RDBMSs only apply to structured data it cannot work with semi-structured or unstructured form of data. To process huge volume of data and analyze it is a big challenge. Proper infrastructure needs to be developed. Some literature [11–13] discusses the difficulties in the development of big data applications. The key challenges are listed as follows [14]: data representation, analytical mechanism, data confidentiality, scalability and optimization

### 1.3.3 Big Data Processing Platforms

As discussed, there are various challenges that need to be addressed for Big Data solutions. For large data storage, management and processing, homogeneous platform cannot be sufficient. The research community has proposed solution from different perspectives. Heterogeneous Distributed Computing (HeDC) framework is utilized to meet the requirement on infrastructure for Big Data, e.g. cost efficiency, elasticity, scalability, storage and management of large datasets. HeDC framework has achieved great success in processing various big data application and accomplishing big data analytics. The physical data center network is the core for supporting big data. Below are the key characteristics for physical data center network.

- **Scalable Computing Infrastructure:** HeDC provides powerful backstage scalable infrastructure support for Big Data processing. HeDC enables distribution and management of Big Data across many nodes and disks.
- **Data Storage Framework:** The big data paradigm has more stringent requirements on storage capacity and processing capacity, as well as network transmission capacity which shall be addressed by HeDC.
- **Parallel / Distributed Programming Framework:** HeDC supports distributed programming for complex computations.
- **Analytics Framework:** HeDC provides analytical platform for processing large volumes of persistent Big Data in highly distributed and efficient manner.

### 1.3.4 Big Data Optimization

For large business enterprises it is indispensable to process large scale data (Big Data) to get better insight into business. There is no doubt that, processing of big data has become challenging task, although many tools and techniques are available to process this flood of data. However, an important concern while processing big data is that its overall performance should not degrade. Increasing rate of data will become critical in the future, so proper optimization techniques need to be applied for Big Data processing.

The challenges discussed for Big Data Processing can be met through HeDC. HeDC provides the solution for scalability, storage and distributed processing. Major research is focused to provide proper optimization techniques to big data sets. Based on the objective, optimization techniques can be categorized as Performance, Ease-of-use and Cost Effectiveness. Performance optimization aims to reduce execution time to make data processing faster. Ease-of-use aims to make data processing tools easier to implement and useable for variety of datasets, while cost effective optimization focuses to minimize the operating cost of the system.

## 1.4 Hadoop

Applications involving Big Data need enormous memory space to load the data and high processing power to execute them. Individually, the traditional computing systems are not sufficient to execute these big data applications but, cumulatively they can be used to meet the needs. This cumulative power for processing Big Data Applications can be achieved by using Distributed Systems with Map-Reduce model under Apache Hadoop framework. Mere implementation of the application on Distributed Systems may not make optimal use of available resources.

### *Hadoop Ecosystem*

Hadoop is open source software comprising of framework of tools. These tools provide support for executing big data applications. Hadoop has very simple architecture. Hadoop 2.0 version primarily consists of three components as shown in fig.4:

1. HDFS (Hadoop Distributed File System) [15]: It provides distributed storage of data over Hadoop environment. It stores data and metadata separately.

2. YARN (Yet Another Resource Negotiator) [16]: YARN is responsible for managing the resources of Hadoop cluster.

3. MapReduce [17]: It is the programming model on top of YARN responsible for processing of data in the Hadoop environment. It performs the computation.

### *A. HDFS*

Hadoop HDFS has master/slave architecture. Master node has two components called Resource Manager and Namenode. Slave on each node of a cluster, is having Node Manager and

Datanode. Namenode and datanode are under the umbrella of the HDFS while Resource Manager and Node Manager are under the umbrella of YARN.

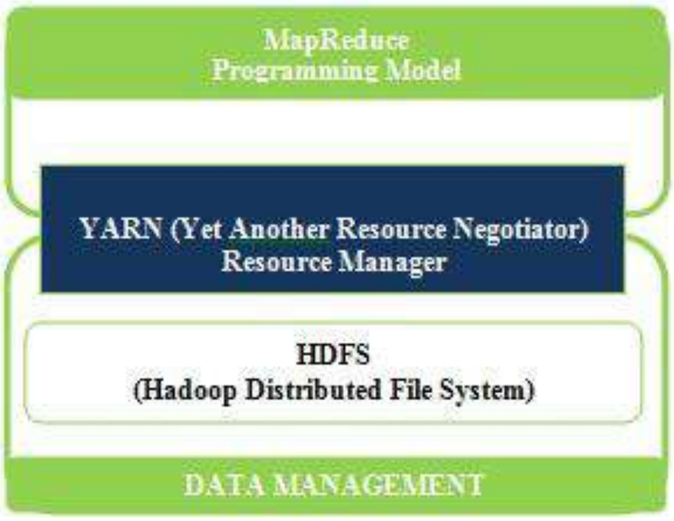

MapReduce
Programming Model
YARN (Yet Another Resource Negotiator)
Resource Manager
HDFS
(Hadoop Distributed File System)
DATA MANAGEMENT

*Figure 4 Hadoop 2.0 Architecture*

The big data applications in Hadoop first assign the task to the master node. Master node will distribute the task among multiple slaves to perform computation and end result will be combined and given back to the master node.

In case of distributed storage, it is important to give indexing for faster and efficient data access. The namenode that resides on the master node is containing the index of data that is residing on different datanodes. Whenever an application requires the data, it contacts the namenode that routes the application to the datanode to obtain the data.

Hardware failures are bound to happen, but Hadoop has been developed with efficient failure-detection model. Hadoop has *de-facto* fault tolerance support for data. By default Hadoop maintains three copies of file on different nodes. Therefore, even in case if one datanode fails, system would not stop running as data would be available on one or more different nodes.

Fault tolerance does not handle the failure of just slave nodes, but it also takes care of failure of master node. There is no single point of failure in case of master node. Hadoop maintains multiple copies of name node on different computer as well as maintains two masters, one as a main master and other as a backup master.

Programmer need not worry about the questions like where the file is located, how to manage failure, how to split computational blocks, how to program for scalability etc. Hadoop implicitly manages all these efficiently. It is scalable and its scalability is linear to the processing speed. In

Hadoop 1.x version, MapReduce was managing both resources and computation. However, Hadoop 2.x splits the two responsibilities into separate entities by introducing YARN.

## *B. YARN*

YARN is a framework to develop and/or execute distributed applications. As shown in fig. 5: Components in the YARN based systems are Global Resource Manager (RM), Application Master (AM) for each application, Node Manager (NM) for each slave node, and an application container for each application running on a Node Manager.

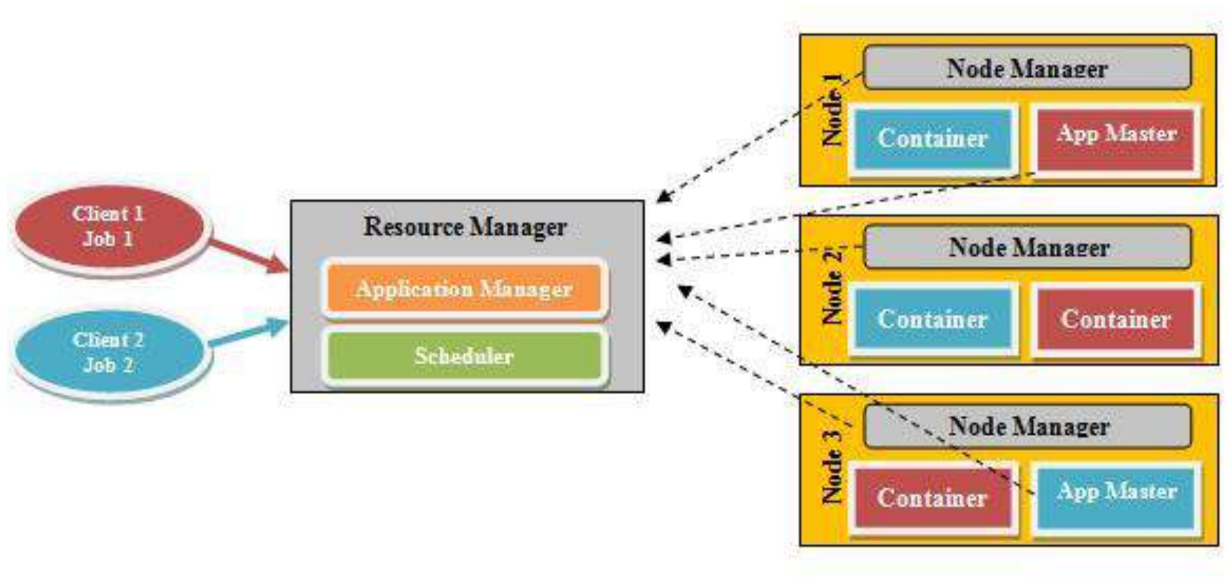


*Figure 5 YARN Architecture*

Resource Manager has two main components: Scheduler and Application Manager. The scheduler schedules the tasks based on availability and requirement of resources. The scheduler schedules the task based on capacity, queues etc. The scheduler allocates the resources by taking consideration of memory, CPU speed, disk capacity etc. The application manager accepts the job from client and negotiates to execute the first container of the application. The application manager provides the failover mechanism to restart the services, which might have failed due to application or hardware failure. Each application manager tracks the status of individual application.

### *C. MapReduce Programming Model*

MapReduce is Google's programming model for processing the colossal amount of data. This model consists of two important phases i.e. maps and reduces. As shown in fig.6 in "map" phase it takes input as key-value (k, V) pair and produces intermediate key-value pair (k1,V1) → {(k2,V2)} as a result while in "reduce" phase it takes a key and a list of the keys and values and generates the final output as key/value (k2; {V2}) →{V3} pair. In distributed processing, it is important to take consideration of data locality. If data to be processed is located near, then it can reduce the time of transmission and can achieve better performance. MapReduce can use this functionality during map-reduce function. In MapReduce each map function will take place on local data and output will be stored to temporary storage.

A master node coordinates the input data only after an input is processed. In the next phase i.e. shuffle phase, it randomly generates values assigned and then sorts it according to the assigned values. Now in reduce phase, it processes the intermediate key-value data and produces the final output.

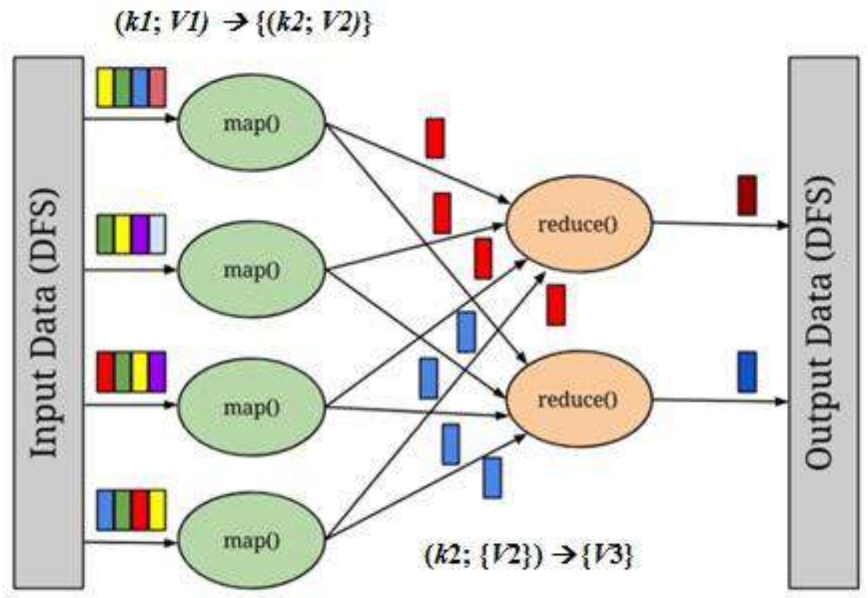


*Figure 6 MapReduce Model*

# Chapter 2: Literature Review

This section literature review is divided in four parts: First, various distributed file systems which are widely used for Big Data storage. Second, various scheduling algorithms for heterogeneous distributed computing are discussed. Third, default schedulers in Hadoop are discussed. Last, alternative approaches to the implementation and improvement of load-balancing algorithms in Hadoop are discussed.

## 2.1 Study of Various Distributed File Systems

Big data implementation is only as good as its file system. From an architectural standpoint, managing the massive volume and throughput of data is a challenge. Big data solutions typically use large, distributed arrays of servers and specialized software. For risk management, a huge amount of data flying across distributed servers also requires exceptional built-in fault tolerance. Various file system for big data storage are: GlusterFS, HDFS, Lustre, Ceph, MooseFS. GlusterFS is a file system from RedHat for its enterprise Linux OS. GlusterFS gets excellent file look-up speed from using elastic hash algorithms rather than centralized metadata. GlusterFS has been most notably used for cloud computing, streaming media, and content delivery. Hadoop HDFS is extremely popular and has gained a great deal of prominence in the big data world. It uses MapReduce as a key function of its data management. Hadoop is an open-source system written in Java designed to run on low-cost hardware. Lustre is a centralized distributed file system which differs from the current DFSs in that it does not provide any copy of data and metadata. Instead, Lustre chooses to store its stores metadata on a shared storage called Metadata Target (MDT) attached to two Metadata Servers (MDS), thus offering an active/passive failover. Ceph is a totally distributed system. Unlike HDFS, to ensure scalability Ceph provides a dynamic distributed metadata management using a metadata cluster (MDS) and stores data and metadata in Object Storage Devices (OSD). MooseFS acts as HDFS. It has a master server managing metadata, several chunk servers storing and replicating data blocks. MooseFS has a little difference since it provides failover between the master server and the meta logger servers.

DFSs are the principle storage solution used by supercomputers, clusters and datacenters. Here, we have given a comparison of four DFSs based on scalability, transparency and fault tolerance. DFSs surveyed are: Lustre, HDFS, Ceph, and GlusterFS. We have seen that the DFSs ensure transparency and fault tolerance using different methods that provide the same results. The main difference lies on the design. In theory, decentralized architectures seem to scale better than a centralized one thanks to the distributed workload management.

Furthermore, the choice of a DFS should be done according to their use. For performance, an asynchronous replication and the use of an index to maintain the namespace are preferable whereas a decentralized architecture is better for managing large amounts of data and requests. The comparison is given in table 1 below.

| | **HDFS** | **Ceph** | **GlusterFS** | **Lustre** |
|---|---|---|---|---|
| **Architecture** | Centralized | Distributed | Decentralized | Centralized |
| **Naming** | Index | CRUSH | EHA | Index |
| **Fault detection** | Fully connected | Fully connected | Detected | Manually |
| **System availability** | No failover | High | High | Failover |
| **Data availability** | Replication | Replication | RAID-like | No |
| **Placement strategy** | Auto | Auto | Manual | No |
| **Replication** | Asynchronous | Synchronous | Synchronous | RAID-like |
| **Load balancing** | Auto | Manual | Manual | No |

*Table 1 Various Distributed File System Comparison*

## 2.2 Scheduling Algorithms in Heterogeneous Distributed Computing

Scheduling in distributed computing system (DCS) is primarily concerned with two aspects such as optimizing completion time of an application and optimizing the resource utilization. In the context of an application, the main parameter is to reduce the total cost of executing a particular application whereas, optimal utilization and performance of the resource is the prime concern of the resource provider. The two main factors in defining the best performance of a scheduling algorithm in HeDCS are application-specific and system-specific. Thus, objective functions of scheduling algorithms can be categorized into two broad classifications: Application-Specific and System-Specific. Figure 7 displays the objective functions of scheduling algorithms covered in this paper.

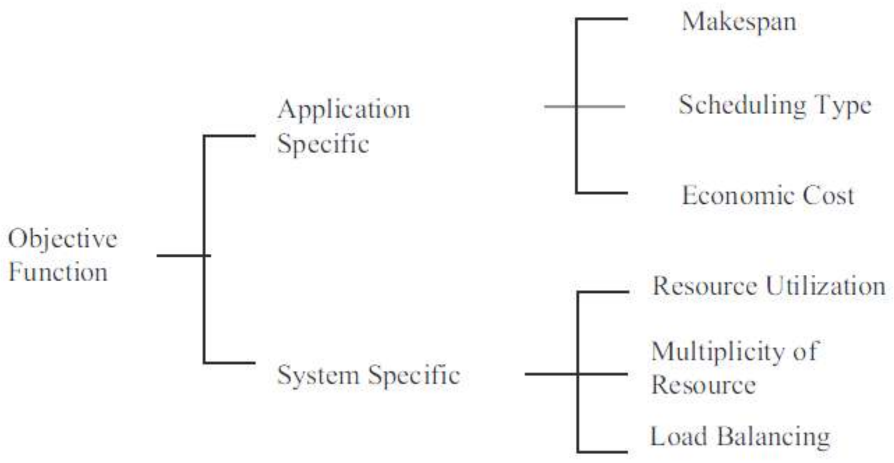


*Figure 7 Scheduling Objective*

### *A. Application-Specific*

Various scheduling parameters need to be considered while implying application-specific scheduling. Application-specific scheduling explicitly addresses heterogeneity and conflict in distributed environments. Watchful scheduling of application components is essential to accomplish its performance objectives. Scheduling decisions are determined based on parameters like application performance, computational requirements, task inter-dependency, processing load and the availability of resources. Depending upon an application, parameters to be considered, may vary, to optimize the performance of a specific application.

### *B. System-Specific*

In system-specific objectives, the main aim is resource utilization, particularly that of processors and memory. The variance in performance of the resources has a direct influence on the performance of the submitted application and must be deliberated during scheduling. Resource utilization, i.e., the percentage of time a resource is busy or available is of vital importance. Overutilization of a scarce resource means non-availability of resource when the application needs it. This may increase the application waiting time, thus resulting in higher completion time. Other resource-specific objectives are load balancing, fixed number of processors, unbounded number of processors, etc. The factors considered in this research paper are scheduling type, multi-core processors, heterogeneity, degree of multiprogramming, makespan, load balancing, multiplicity of resources, impact of bounded number of processors (BNP) and unbounded number of heterogeneous processors, optimized resource time, etc.

### 2.1.1. *Comparative Study of Scheduling Algorithms*

The study emphasizes on two aspects, one to find the objective behind using specific scheduling technique, and secondly discussed the merits and possible enhancements to each technique. In Table 2, various algorithms have been listed.

**Zheng et al. [18] [2013]** proposed Monte Carlo based Directed Acyclic Graph scheduling approach with the objective to minimize the makespan for BNP. This approach works well for any random distribution under heterogeneous environment. This approach gives competitive advantage compared to other static heuristic techniques.

**Ehsan et al. [19] [2013]** proposed Stand deviation-based algorithm for task scheduling (SDBATS) to reduce schedule length and speedup the scheduling by assigning task priority.

**Kwok et al. [20] [1999]** proposed to optimize the makespan by considering a wide range of techniques, genetic algorithm, randomization branch-and- bound and graph theory. Authors have proposed many useful static, heuristic algorithms (e.g. HEFT, MCP, ETF, and DLS) but that won't the effective in today's era of big data.

**Kanemitsu et al. [21] [2016]** proposed clustering based task scheduling algorithm that minimizes the schedule length for heterogeneous processors. It is apt for data intensive application and has proven to be better than other list-based and clustering-based task scheduling algorithms.

**Abdelkader et al. [22] [2012]** proposed dynamic task scheduling algorithm for heterogeneous systems called Clustering Based HEFT with Duplication (CBHD). This algorithm targets the three important parameters for getting better performance, minimize the makespan, load balancing and optimize the sleek time.

**Wang et al. [23] [2016]** proposed Heterogeneous Scheduling algorithm with improved task Priority (HSIP) for improvising schedule length ratio and task priority. This algorithm performs two-step process first, identifies the task priority and second finds the best processor to execute the tasks.

**Ahmad et al. [24] [2012]** proposed Performance Effective Genetic Algorithm (PEGA) which operates through large search space and finds the best solution using reproduction concept.

Reproduction uses two operators namely crossover and mutation to select a random task and performs fitness function on it to select the best task to execute on the heterogeneous parallel multiprocessor system.

**Ahmad et al. [25] [2016]** proposed Hybrid Genetic Algorithm (HGA) is a hybrid combination of HEFT heuristic and PEGA genetic algorithm. It provides optimize makespan and load balancing over heterogeneous systems.

**Valeria, et al. [26] [2015]** proposed Distributed QoS-Aware Scheduling with self-adaptive capability in storm. By using this concept authors tried to overcome the limitation of high latency, less availability and poor system utilization in distributed data stream processing (DSP).

**Hamid et al. [27] [2013]** proposed Predict Early Finish Time (PEFT) to speedup and optimize the makespan. It has two phases: a task prioritizing and a processor selection which identifies the task priority and allocates it to the best processor respectively.

**Khaldi et al. [28] [2015]** proposed static-heuristic scheduler called Bounded Dominant Sequence Clustering (BDSC) is an extension of DSC limiting the memory constraints and the bounded number of processors. It is suitable for signal processing and image processing kind of application.

**Kenli Li et al. [29] [2015]** proposed stochastic dynamic level scheduling (SDLS) algorithm to minimize the makespan. This algorithm outperforms when tasks arrive randomly.

**Jorge et al. [30] [2011]** proposed Parallel Heterogeneous Earliest Finish Time (P-HEFT) which is an extension to HEFT. P-HEFT supports parallel task DAG which provides optimized makespan that makes it suitable for image processing type of application.

**Pravanjan Choudhury et al. [31] [2012]** proposed online scheduling of dynamic task graphs. Algorithm provides dynamic path selection option by scheduling tasks at run time. The proposed algorithm is assumed to be limited to homogeneous systems. But it can be extended further to heterogeneous systems by taking the base of this algorithm.

**Tang, Z. et al. [32] [2015]** proposed Self-Adaptive Reduce Scheduling (SARS) for Hadoop platform. During MapReduce phase, it reduces the waiting time by selecting an adaptive time to schedule the reduce task. This method reduces the turnaround time.

**Yuxiong et al. [33] [2011]** proposed Multi-Queue Balancing (MQB) algorithm that minimizes the makespan and maximize the heterogeneous resource utilization. MQB has multiple queues for online scheduling to achieve better utilization and minimizing completion time.

*Table 2. Comparison of Various Scheduling Algorithms*

| Sr. No | Algorithm | Key Parameter | Scheduling Type | Nature of Task | Environment | Objective | Comments | Algorithm compared |
|---|---|---|---|---|---|---|---|---|
| **1** | Monte Carlo based DAG* scheduling approach [18] | Makespan, Heterogeneity, Bounded Number of Processors | Global, Static, Sub-optimal, Heuristic | Flow of Work | Dynamic DAG Simulator | To minimize the makepan and optimize overall performance | Avoid the complex computation with random variable and applicable to any random distribution.DAG HEFT based which is less effective. | - |
| **2** | SD*-Based Algo. for Task Scheduling - SDBATS [19] | Makespan, Multi-core processors | Global, Static | Periodic Task | Not mentioned | Effective Schedule length and speed up | Suitable for some real-world application like Gaussian elimination and Fourier transformation | - |
| **3** | HEFT*, MCP*, ETF*, HLEFT*, DLS* [20] | Makespan, Degree of multiprogrammi | Global, Static, Sub-optimal, Heuristic | Simultaneou s Tasks | Bounded No. of Homogeneous Processors | To minimize make span | Standard algorithms for static task scheduling. | - |
| **4** | Clustering for Minimizing Schedule Length - [21] | Schedule length | Global, Static | Task Clustering | Not mentioned | To minimize Worst Schedule Length (WSL) | Suitable to data intensive application and heterogeneous system support | Clustering based Task Scheduling |
| **5** | Cluster-ing Based HEFT with Dupli-cation – CBHD [22] | Makespan, Load Balancing, Optimize sleek time | Global, Dynamic, Distributed | Grouped Task (Cluster) | Distributed Algorithm Simulator | To minimize the execution time & provide load balancing. | Minimize makespan, maximize processor utilization by load balancing. | HEFT, Triplet Cluster |
| **6** | Multi Queue Balancing - MQB [33] | Makespan, Load Balancing | Global, Dynamic, Distributed | Parallel Tree Workload based Structure | Discrete- time Simulator | Reduce exec. time by max. resource utilization | Reduces the processor idle time and full use of resources | Compared with various types of load. |
| **7** | Heterogeneous Scheduling Algorithm with improved Task Priority – HSIP [23] | Schedule length ration, efficiency | Global, Static | Simultaneou s Task | Heterogeneous Simulation | Improve task priority strategy for optimized makespan | Algorithms well suitable for heterogeneous environment and useful for real world problems | PEFT, SDBATS, HEFT, CPOP* |
| **8** | Performance Effective Genetic Algorithm - PEGA [24] | Optimal mapping, sequence of execution | Global, Static Optimal | Random Task | Heterogeneous Simulation | To minimize finish time along with increase throughput | Algorithm outperforms against traditional scheduling methods. Not suitable for large data sets | RR*, SJF*, FCFS* |

| Sr. No | Algorithm | Key Parameter | Scheduling Type | Nature of Task | Environment | Objective | Comments | Algorithm compared |
|---|---|---|---|---|---|---|---|---|
| **9** | Hybrid Genetic Algorithm – HGA [25] | Makespan, Load Balancing | Global, Static Optimal | Regular/ Random Task | Heterogeneous Simulation | Optimization of makespan and utilize maximum resources | Follows Montage, CyberShake benchmark for validate performance | MCP, HEFT, PEGA,MPQGA*,HSCGS* |
| **10** | Distributed QoS aware scheduler [26] | Latency, availability, sys. utiliz. | Global, Dynamic, Adaptive | Distributed Stream Processin | Peersim Simulator | Improve perf. by adaptive scheduler for distributed | Storm based scheduler for overall system perf. and quality of services. | cRR*, cOpt* |
| **11** | Predict Earliest Finish Time – PEFT [27] | Efficiency, Makespan, better schedule | Global, Static | Workflow | Not mentioned | Forecast the cost table and schedule the task accordingly to | Suitable when all tasks are known along with their complexity and dependency. | Look-ahead, HEFT,HCPT*, PETS*, HPS* |
| **12** | BDSC Hierarchical BDSC - HBDS [28] | Resource Management, Speedup, Bounded | Global, Static, Sub-Optimal, | Parallel Task Execution | PIPS Platform | Effective resource management and speedups on shared and | Suitable for signal processing, image processing application. For BNP system is | HLFET, ISH, MCP, HEFT |
| **13** | Stochastic Dy-namic Level Sched-uling - (SDLS) [29] | Makespan, Speedup, and makespan standard | Global, Dynamic, Distributed | Precedence constrained tasks | Simulation cluster | To achieve better performance by dynamic scheduling | Effective for random task arrival time. This will be more effective by parallelism. | HEFT, Rob-HEFT, and SHEFT |
| **14** | P-HEFT [30] | Makespan, Heterogeneity Parallel task execution | Global, Dynamic | Simultaneous Tasks | Not mentioned | To minimize the completion time by parallel execution. | Suitable for image processing application. Best for multiple-mixed jobs. | Heterogeneous Parallel Task Scheduler (HPTS) |
| **15** | Online Scheduling of Dynamic Task Graphs [31] | Number of Processors, Memory | Global, Dynamic | Real time Tasks | Not mentioned | To provide runtime scheduling | Interprocessor communications of fixed number of homogeneous processors. | Multiprocessor scheduling algorithm |
| **16** | Self Adaptive Reduce Scheduling – SARS [32] | Reduce completion time | Global, Dynamic, Distributed, Optimal | Batch processing | Hadoop Map-Reduce Cluster | To minimize average completion time and response time | Suitable to big data application processing. | FIFO, Fair , Ca-pacity Scheduler |

*****DAG- Directed Acyclic Graph, SD-Stand Deviation, HEFT- Heterogeneous Earliest Finish Time, MCP- Modified Critical Path, HLEFT- Highest Level First Estimate Time, DLS-Dynamic Level Scheduling, GA- Genetic Algorithm, ETF-Earliest Time First, ISH-Insertion Scheduling Heuristic, HCPT-Heterogeneous Critical Parent Trees, PETS-Performance Effective Task Scheduling, HPS-High Performance Task Scheduling. SHCP-Scheduling with Heterogeneity using Critical Path, HHS-Hybrid Heuristic Scheduling, RR-Round Robin, SJF, Shortest Job First, FCFS - First Come First Serve, BDSC- Bounded Dominant Sequence Clustering, MPQGA-Multiple Priority Queues Genetic Algorithm, HSCGS-Hybrid Successor Concerned Heuristic-Genetic Scheduling, cRR-Centralized Round-Robin, cOpt-Centralized Optimal scheduler, FIFO-First In First Out

## 2.3 Study of Scheduling Algorithms in Hadoop

Hadoop supports three scheduling schemes in MapReduce framework: FIFO, Capacity [34] and Fair [35] scheduler. MapReduce1 (MR1) comes with all three with FIFO as default scheduler, while MR2 comes with capacity and fair scheduler, which can be further configured with delay scheduler to address the locality issue.

***A. Capacity Scheduler***

This is the default scheduler, which comes with the MR2 or YARN. The capacity scheduler's configuration supports multiple queues, which can be allocated to multiple users based upon tasks or organization. This scheduler is designed with an idea that same cluster can be rented to multiple organization and resources may be divided to several users. Thus, the organization can divide their resources across multiple departments or users depending upon their tasks or the cluster can also be divided among multiple subsidiary organization. Each queue can be configured with fix portion of resources, which can be soft or hard. Generally, resources are soft having elastic allocation, but can also be configured for hard approach.

Capacity scheduler makes use of FIFO (First-In First-Out) scheduling if multiple jobs are in the same queue. Suppose a job comes into the queue "A" and if queue "A" is empty, then it allocates all the resources to the first job. This would utilize more resources then configured capacity of queue, particularly if queue allocation is elastic and job requires more resources. When a new job comes in queue "B", assuming that the first job is still running and using the resources more then it's allocated capacity, then tasks of first job will be killed to free up the resources and allocate that resources to second job. Suppose if another job comes to the queue "A" or "B" the capacity scheduler will process it like FIFO or FIFO with priority. There are many features available like: capacity guarantee, elasticity, security etc. that can be customized as per requirement.

***B. Fair Scheduler***

Fair schedulers have similar queue configuration as discussed in capacity scheduler. Jobs would be submitted to the queue, which is termed as a "pool" in case of fair scheduler. Each job will use the allocated resources to their pools. As in capacity scheduler, FIFO approach is followed the jobs which are coming late has to wait till the time first job finishes or resources made available, so this problem is solved in the fair scheduler that the jobs which have waited in the queue would be picked up and would be processed in parallel with the same amount of resources shared by the applications which are in the same queue. Fair scheduler support three scheduling policy that is: FIFO, Fair, and DRF (Dominant Resource Fairness).

In FAIR-FIFO scheduling policy, if multiple jobs are in the same queue then resources will be allocated to the job, which enters first in the queue, and each job will run serially. However, fair sharing is still being done between the queues.

In FAIR-FAIR scheduling policy, the fair amount of resources will be shared by the jobs that are running in the same queue.

FAIR-DRF scheduling policy is devised by Ghodsi. In FAIR-DRF scheduling policy, DRF evaluates the resources shared by each user, finds out the maximum of it, and calls it as a dominant resource of the user. The idea is to make uniform resource sharing among the users through equalizing the resources like CPU and Memory.

## 2.4 Study of Performance Improvement Algorithms in Hadoop

To improve performance of Hadoop many researchers have work on many diversified areas. In distributed computing load balancing is the key area which affects significantly in terms of overall performance as you're working with thousands of computer in clusters. Many researchers have worked on performance improvement through effective load balancing using various custom designed algorithms and programming models. In paper [36] authors have summarized notable research contribution for load balancing by scheduling [37,38], load balancing during job processing [39,40] and load balancing using custom block placement [41-44]. In this section we summarize some of the noteworthy work done to achieve better performance in Hadoop using load balancing and custom block placement strategy.

In paper [45] authors propose the approach which places the blocks based on region placement policy. Data is stored into plurality of regions rather than plurality of nodes. Therefore, complete replica of the region can be stored in a contiguous portion of data. This policy achieves great fault-tolerance and data locality for region-based cluster storage. Authors of paper [46] propose heterogeneous storage media aware strategy which collects storage media, processing capacity and stores them on different storage media types (i.e. HDD, SSD, RAM) according to workload balance. Experiment proves that it reduces imblancing of cluster. In paper [47] authors propose dynamic replica placement which works on Markov probability model and places replica homogeneously across the racks. Results shows better job completion time compare to HDFS and CDRM and also distribute the replica uniformly across all the nodes. In paper [48] authors propose strategy which considers network load and disk utilization for placing data blocks. Proposed strategy outperforms default and real-time block placement policy and achieves better performance in terms of throughput and storage space utilization. In paper [49] authors propose improved slot replica placement policy which considers heterogeneity of nodes and partitions all nodes in 4 sections to store data blocks. Section wise partition scheme achieves greater load balancing and eliminates the use of HDFS balancer. In paper [50] authors propose strategy which tracks spatial characteristics of data to co-locate them. If data blocks are geographically distributed across multiple data centres without concern where job is running then it degrades the performance tremendously. Here authors have achieved better query execution time by adding spatial data awareness which effectively reduces the job execution time. In paper [51] authors propose probability based DLMT (Data Local Map Task Slot) approach which adjusts the data placement rate in along with replica eviction policy to improve Hadoop performance and cluster space utilization respectively. In paper [52] authors propose a model called “Starfish” which dynamically adjusts the Hadoop parameter according to workload of job. Starfish work with each phase of Hadoop, starting with job level tuning, real time parameter adjustment and finally process scheduling. It achieves great performance compare to default Hadoop setup, placement policy and scheduling scheme. Below table 3 summarizes the work done by the researchers.

| | **Performance Improvisation Factors** | **Research Contribution** | **Remarks** |
|---|---|---|---|
| A region-based placement policy | Fault-tolerance and data locality | Designed region based cluster storage system which stored once complete replica of the region on single node | This scheme is helpful when plurality of region servers is required. |
| Robust Data Placement Scheme (RDP) | Load balancing and optimal network congestion | Proposed RDP scheme considers the storage type (i.e. SSD, HDD and RAM) and processing speed of node for balancing. | Authors have successfully demonstrate how storage type and computing capacity prediction can achieve better load balancing and reduce network overhead. Pre-processing for RDP scheme takes significant amount of time when multiple clusters with variety of nodes are there. |
| Dynamic Replication Strategy (DRS) | Job scheduling time and disk utilization rate | Proposed dynamic replica placement based on Markov model. | Authors have successfully tested model on homogeneous cluster. Authors have not considered the time for replication adjustment which is important justification. |

| | | | |
|---|---|---|---|
| Network sensitive strategy | Strong fault-tolerance block placement and high throughput | Designed scheme which considers network load for data placement. Try to place replica on low network loaded group of nodes. | Proposed strategy reduces the inter-rack transfers which eventually increase the performance also works with heterogeneous cluster. Authors have not considered the load imbalancing issue in Hadoop. |
| Improved replica placement policy | Load balancing | Designed policy which evenly distributes the replicas into section. | Proposed policy achieves even load balancing across nodes which eliminates the use of HDFS balancer. Policy only proposed for homogeneous cluster. |
| CoS*-HDFS | Reduce total execution time and network bandwidth. | Proposed algorithm which is aware of geo-spatial data blocks. | Proposed algorithm improves performance of MapReduce query execution and reduces network traffic. |
| Data Replication Method | Data locality and replication method | Proposed LRFA* policy effectively uses storage space of cluster to achieve better data locality. | Effectiveness and performance is not evaluated which they've claimed. |
| Starfish | Self-tuning approach | Proposed self-tuning Hadoop model to achieve better performance. | Improved block placement policy significantly improves job running time. Dynamic tuning also tested successfully. |

*CoS- Co-Locating Geo-Distributed Spatial Data, LRFA- Least Recently Frequently Access

*Table 3 Performance Improvement Related Research Contribution*

# Chapter 3: Proposed Model

To achieve to better performance for big data processing we target upon two important aspects of heterogeneous distributed computing: file system management and process management.

First, file system management basically controls the block placement and allows us to rearrange the blocks to specified nodes based upon two important approaches of load balancing:

1. Balance the load among heterogeneous and homogeneous nodes of the cluster.
2. Balance the load within the cluster based on the processing capability of each node by giving priority to each node.

Our proposed algorithm allows the user to select the nodes based on their individual data processing capability and rearrange the blocks which are placed using the default policy. Default block placement policy of HDFS fails to achieve optimized performance as it does not check the processing capability of the node while placing blocks to that node. Our proposed algorithm achieves that by considering processing capability of nodes and places blocks which has higher processing capability and if nodes are having same processing capacity then it also checks its utilization, and considers the less utilized node first, for block placement. For processing capability assignment we use static priority assignment in config.xml file. We have assigned priority-2 to the nodes where we want to rearrange all blocks and priority-1 to the nodes which won't contain any blocks. By doing this we have control over data to be put on selected nodes, considering processing capacity and utilization of nodes.

Second, we use the concept of node labeling to achieve better process management. YARN Node label [53] allows partitioning the single cluster among multiple sub-clusters. Using this concept we can mark nodes with meaningful labels i.e. Nodes with higher processing capability may be labeled as "high_cpu" and with high memory may be labeled as "high_mem". By combining the proposed algorithm with node label, we can actually select where to put jobs. To achieve better data locality we put our job to the nodes where data is actually rearranged by our algorithm. At last, we use Hadoop scheduling to put jobs in queues for job processing. This also limits the overhead of internode and inter-rack data transfer, since process (containers) and data blocks are on to the same nodes. Figure 8 shows our proposed model.

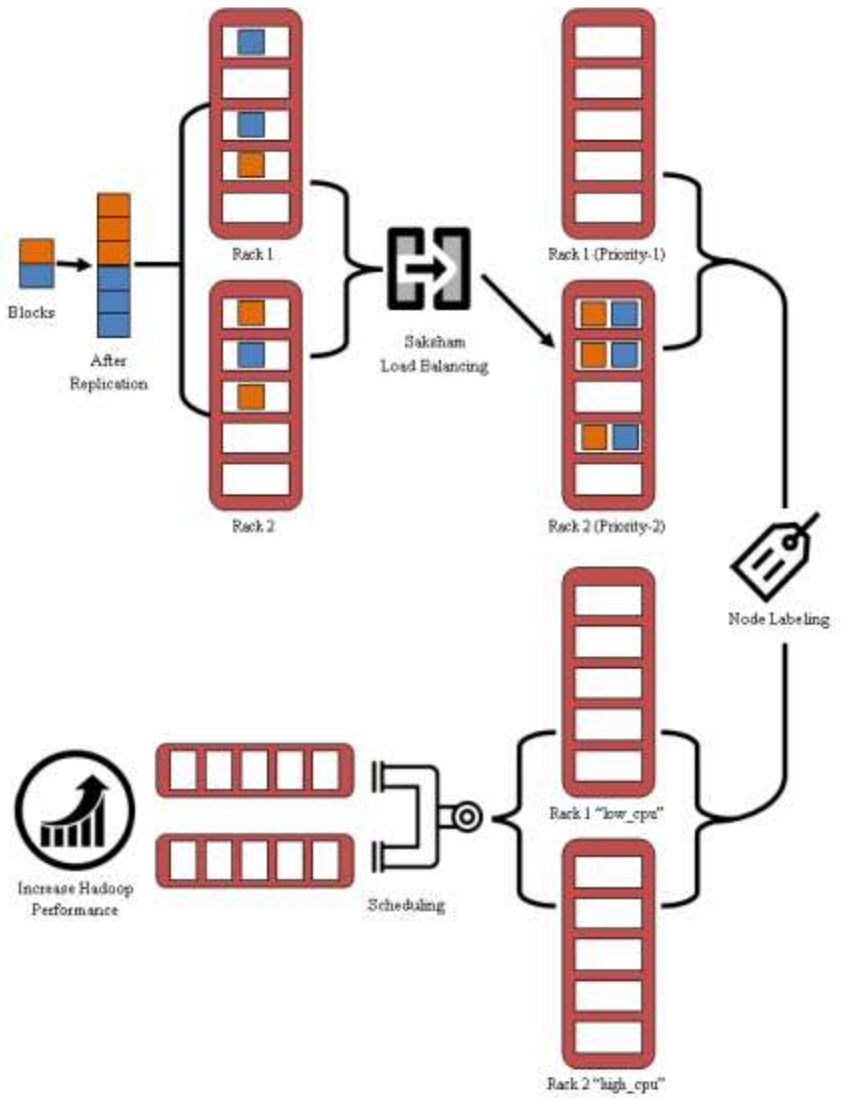


*Figure 8 "Saksham" Model*

## 3.1 Saksham: Block Rearrangement Algorithm

Hadoop uses "Rack Awareness" while placing data blocks for fault tolerance and to achieve better performance. "Rack Awareness" is a concept Hadoop uses to place read/write request to the same rack or nearby rack [54]. This concept helps to achieve better data locality as discussed in section 2. MapReduce *de facto* standard tries to move the job where data is stored. But, that node may not have sufficient processing capability or job may get skewed due to less processing/memory capability. Hence, we propose "Saksham: Resource Aware" algorithm which rearranges the data blocks according to user defined processing capability or heterogeneity of environment.

We can apply custom block rearrangement policy by considering two distinct ways. First, we have heterogeneous nodes with different computational capability. Second, we can assign two separate groups for processing depending upon needs of application, homogeneous and heterogeneous nodes. Initially default HDFS block placement policy places data blocks as shown in fig. 9. Figure 9 shows how data placement would place 8 blocks, if client requests from node1 considering replication factor 3.

For the first way, we will consider processing capability of nodes for forming a group. For this case, we have considered heterogeneous nodes so nodes are having different processing capability. Few nodes are comparatively slower which may degrade the overall performance of processing. We've divided nodes into two groups priority=1 and priority=2. Assigned priority=2 for nodes which have more processing capability and priority=1 which has lower processing capability. These settings will take place in config.xml file. Once priority is set our "Saksham" algorithm will rearrange the blocks of specified HDFS file path and store all blocks onto the nodes which have priority=2 and remove all the blocks from nodes which has priority=1.

For the second way, we will formulate the group of homogeneous and heterogeneous nodes. We can assign priority 1 or 2 to either of the group. Depends upon big data processing application need we can rearrange all blocks to heterogeneous nodes or homogeneous nodes only. Only nodes with priority=2 will store the blocks and with priority=1 will not store any of them.

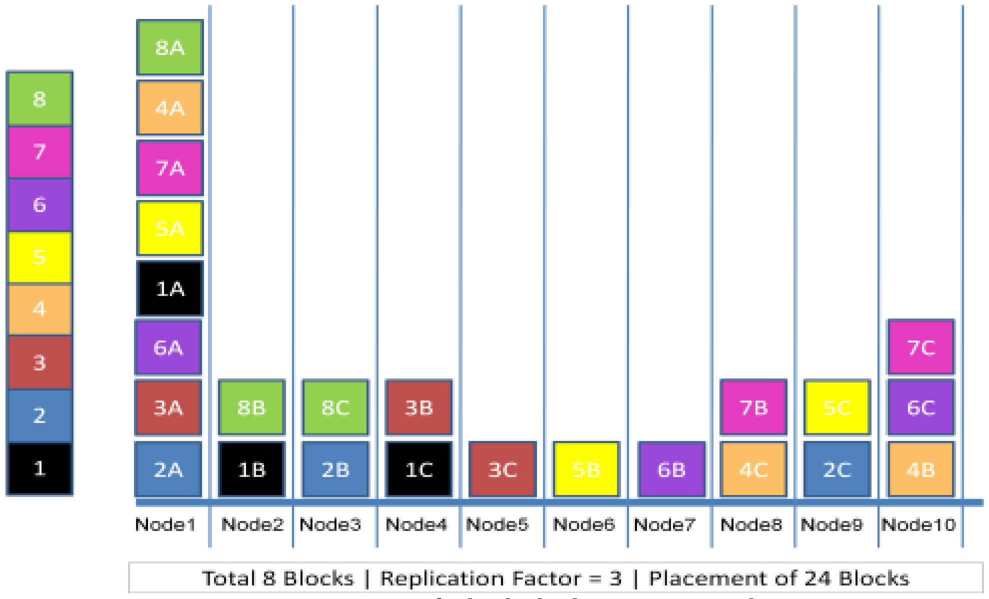


*Figure 9 HDFS Default Block Placement Example*

While placing all blocks our proposed algorithm also take care that blocks of a file will distribute equally over all the nodes by considering disk usage of each node. Each time before placing the block of file it checks that node should not contain a replica of the same block. The fact that Hadoop works better for the homogeneous environment, fulfilled by our strategy even though cluster is heterogeneous. Figure 10 demonstrates how our proposed "Saksham: Resource Aware Block Rearrangement" policy can rearrange blocks according to a priority assigned.

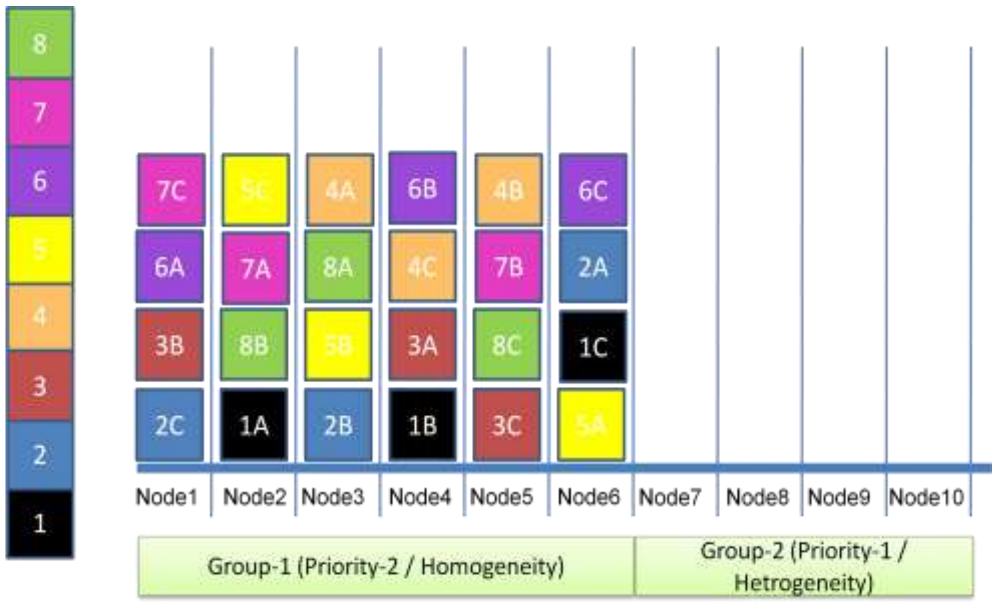


*Figure 10 "Saksham" Approach: Group-1: Priority=2; Group-2: Priority=1*

Below fig.11 is the proposed "Sakasham: Resource Aware Block Rearrangement" algorithm.

- Firstly, algorithm checks the HDFS path contains blocks and also collects the list of datanodes from the DatanodeInfo, Hadoop API and stored them in arraylist.
- Secondly, based upon statically configured config.xml file it checks the priority assigned to each node and segregates the nodes in two lists i.e node_list1 (priority-1) and node_list2 (priority-2).
- In last step blocks along with each replica will be rearranged in nodes which are having priority-2. For rearrangement, it checks the disk utilization and processing capability of each node.

**Algorithm 1** Saksham: Resource Aware Block Rearrangement algorithm

```
Input: HDFS location of input files to be balanced / rearranged
File contains list of data blocks which are placed in HDFS using default policy.
Output: Data blocks will be placed to specific nodes only based on given priority factor.
1)  if input HDFS path != null
2)      foreach locatedBlocks block : nameNode.getLocatedBlocks() do
3)          Put blocks in arraylist<block_list>
4)      endfor
5)  endif
6)  foreach DatanodeInfo node : getDatanodeStats(Live) do
7)      if nodes in config.xml != null
8)          Add nodes in arraylist<datanodes>
9)      endif
10) endfor
11) foreach BalancerDataNode node : datanodes do
12)     if nodes.priority = 1.0
13)         Add nodes in arraylist<node_list1>
14)     elseif nodes.priority = 2.0
15)         Add nodes in arraylist<node_list2>
16)     endif
17) endfor
18) Sort node_list2 by disk utilization in ascending order
19) for each block replica do
20)     Initialize <block_list>  queue with all blocks
21)     Initialize <node_list2>  nodes with priority=2
22)     for each block replica in <block_list> do
23)         if (find first node form <node_list2>) doesn't contains(block)
24)             Put block onto  selected node
25)             Remove node from <node_list2>
26)             if node_list2 is empty
27)                 Initialize with all nodes with priority=2
28)             endif
29)         endif
30)     endfor
31) endfor
```

Figure 11  Proposed “Saksham” algorithm

# Chapter 4: Results

## 4.1 Experiment Setup

We have tested our experiment on Grid'5000 [55] heterogeneous cluster. Grid'5000 is large-scale distributed testbed for the researchers to experiment their research on high configurable cluster. We have used 10 nodes for our experiment. The cluster is configured for Hadoop 2.7.2 version. The configuration of nodes is shown in below table 4. Table 4 shows the heterogeneity of nodes in terms of CPU, memory, storage, number of cores and networks. Table 5 shows priority and node label settings for block rearrangement and the job processing respectively.

| CPU | Detail Specifications | No of nodes |
|---|---|---|
| Intel Xeon E5-2630 v3 | **CPU:** 2 CPUs/node **Cores:** 8 cores/CPU **Memory:** 128 GB memory **Storage:** 558 GB/node, 10 **Network:** 10 Gbps | Parasilo-[1-6] Total - 6 |
| Intel Xeon X5570 | **CPU:** 2 CPUs/node **Cores:** 4 cores/CPU **Memory:** 24 GB memory **Storage:** 465 GB/node, 10 **Network:** 20 Gbps | Parapide-[1-4] Total - 4 |

*Table 4 Hadoop 2.7.2 heterogeneous cluster configuration*

| Priority-2 | Priority-1 |
|---|---|
| **Node Label – "high_cpu"** | **Node Label – "low_cpu"** |
| parasilo-1.rennes.grid5000.fr | parapide-1.rennes.grid5000.fr |
| parasilo-2.rennes.grid5000.fr | parapide-2.rennes.grid5000.fr |
| parasilo-3.rennes.grid5000.fr | parapide-3.rennes.grid5000.fr |
| parasilo-4.rennes.grid5000.fr | parapide-4.rennes.grid5000.fr |
| parasilo-5.rennes.grid5000.fr | |
| parasilo-6.rennes.grid5000.fr | |

*Table 5 Data placement priority and node label settings*

## 4.1 Experiment Results

Initially, we have placed blocks using the default HDFS block placement policy. Afterwards, we have applied "Saksham" block rearrangement policy according to settings described in table 4. We have tested our "Saksham" algorithm using 2 datasets of different sizes. First, "Bag of words" static text data set of 5 GB and 10 GB. Second 5 GB and 10 GB data generated using TeraGen utility. TeraGen generates random data that can be conveniently used as input data for a subsequent TeraSort run. Text data set is placed using the default HDFS placement policy while TeraGen generates data using MapReduce and places accordingly. In both, the case replication factor is 3 so total block size for rearrangement is 15 GB and 30 GB. Figure 12 [A-B] and fig. 13 [A-B] shows how blocks are rearranged from default placement to our selected nodes only (i.e. parasilo-[1-6]) with priority is set to 2.

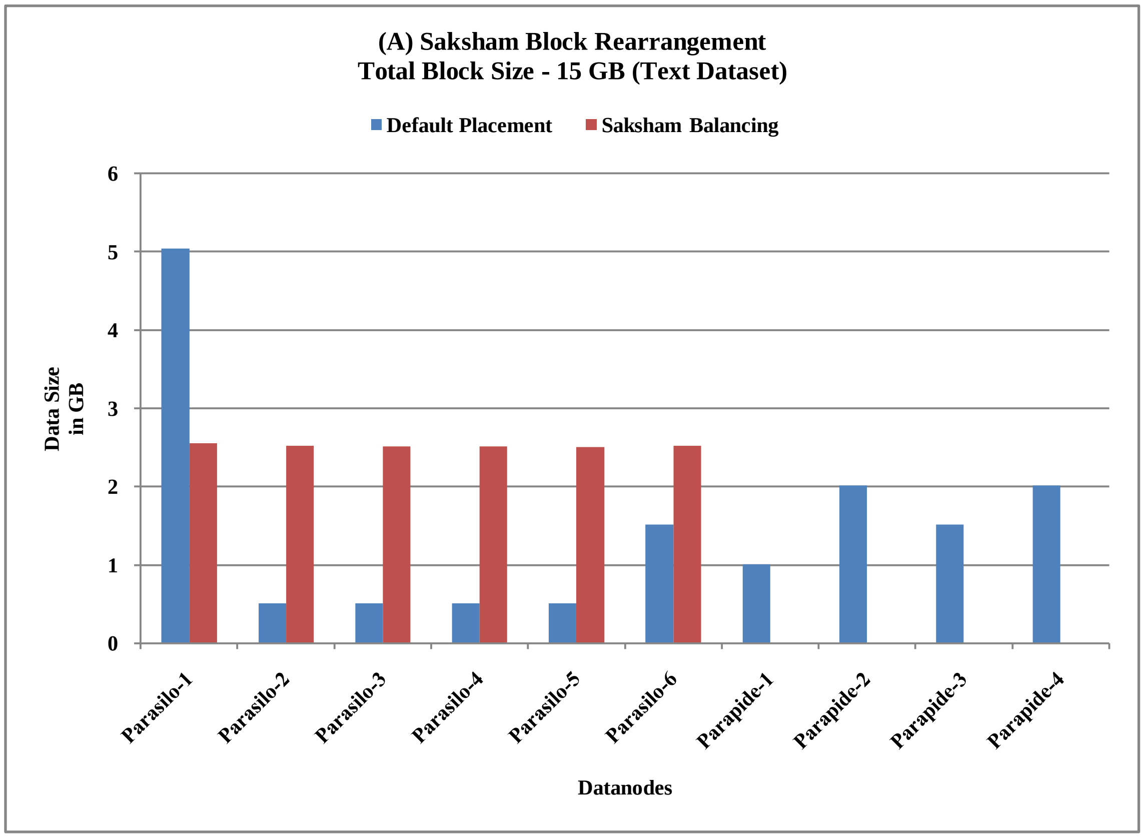


*Figure 12 [A] Saksham Balancing: Text dataset (A) Size-15 GB*

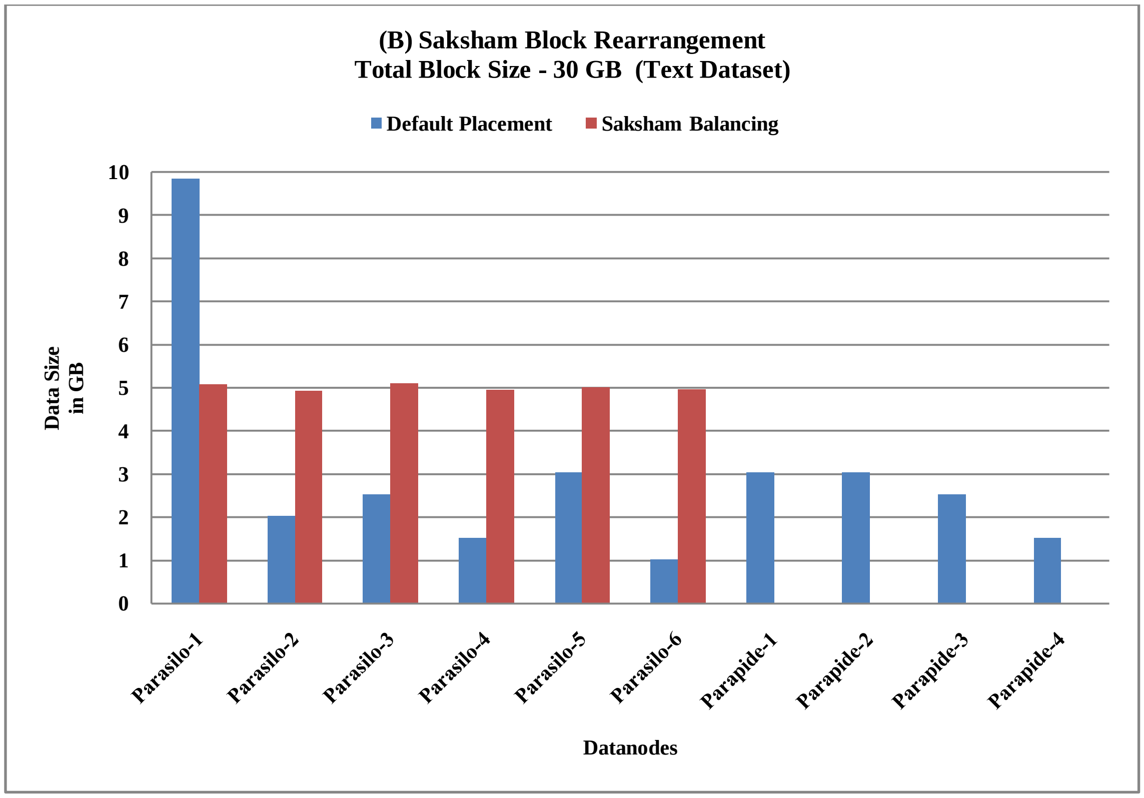


*Figure 132 [B]  Saksham Balancing: Text dataset (B) Size-30 GB*

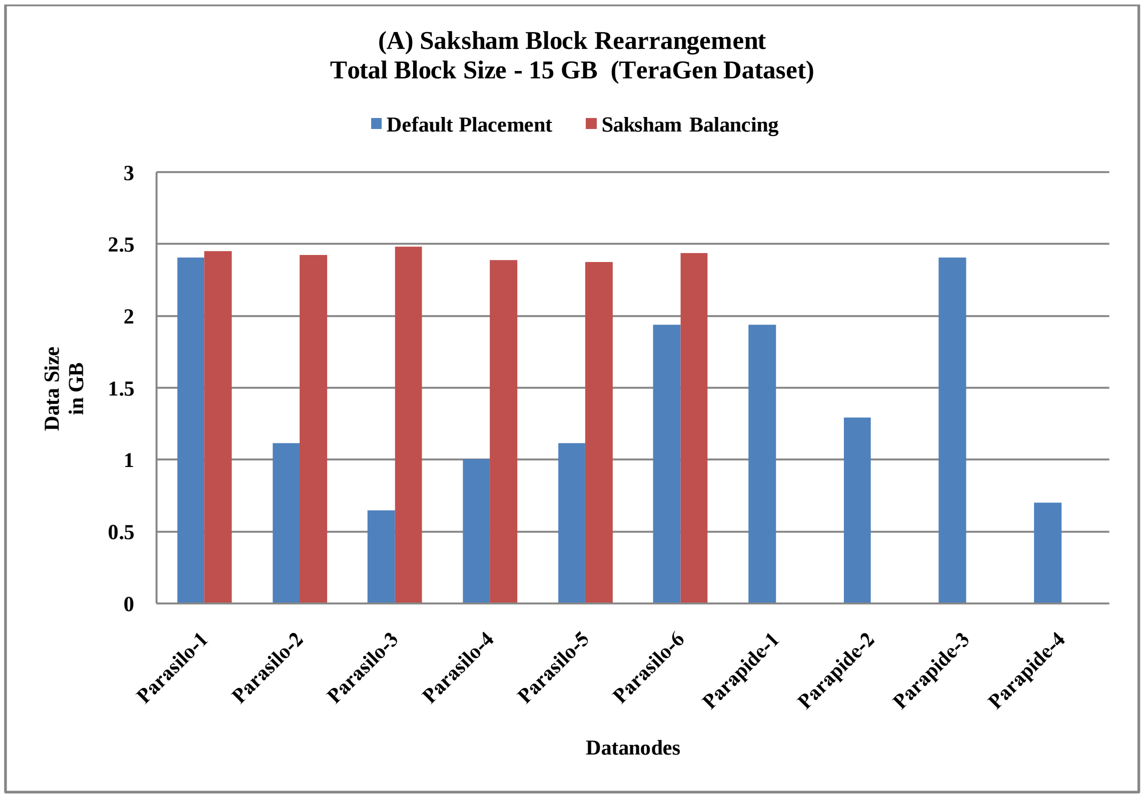


*Figure 13[A]  Saksham Balancing: TeraGen dataset (A) Size- 15 GB*

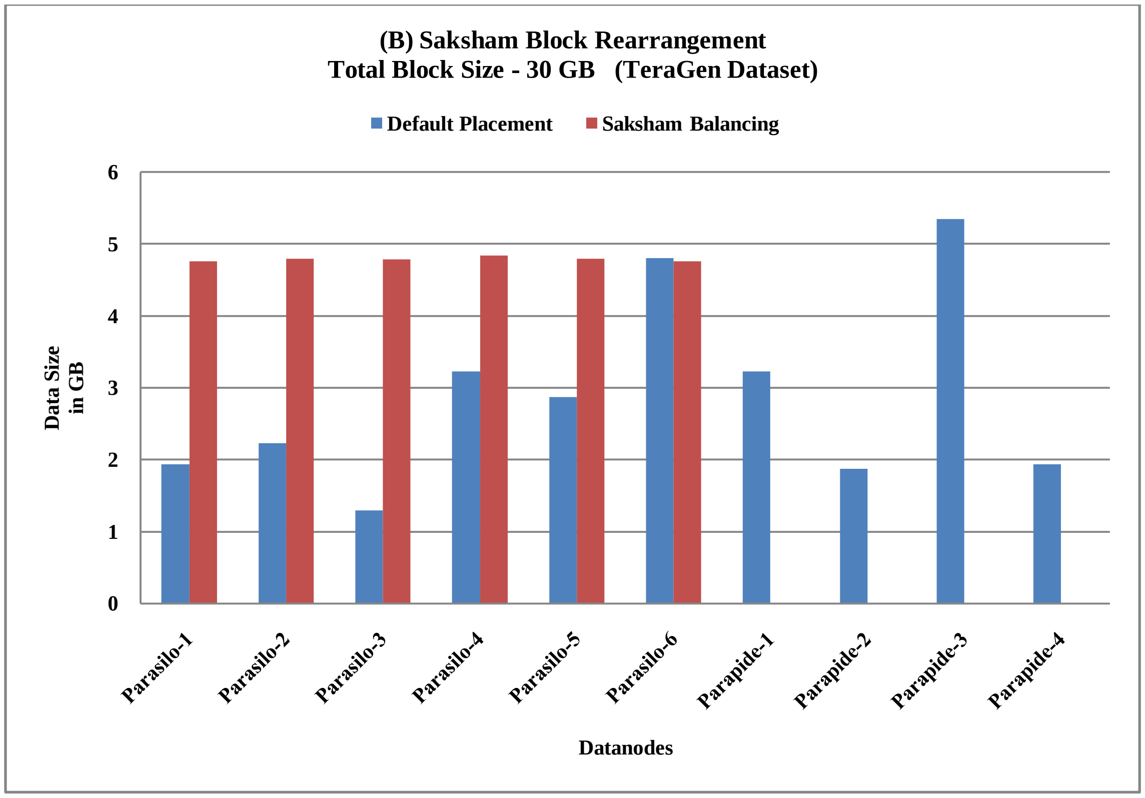


*Figure 13[B] Saksham Balancing: TeraGen dataset (B) Size- 30 GB*

| | Text Data 15 GB | | Text Data 30 GB | | TeraGen 15 GB | | TeraGen 30 GB | |
|---|---|---|---|---|---|---|---|---|
| **Datanodes** | **Default Placement** | **Saksham Balancing** | **Default Placement** | **Saksham Balancing** | **Default Placement** | **Saksham Balancing** | **Default Placement** | **Saksham Balancing** |
| Parasilo-1 | 5.04 | 2.55 | 9.85 | 5.08 | 2.41 | 2.45 | 1.94 | 4.77 |
| Parasilo-2 | 0.50 | 2.53 | 2.02 | 4.93 | 1.12 | 2.43 | 2.23 | 4.80 |
| Parasilo-3 | 0.50 | 2.51 | 2.52 | 5.09 | 0.65 | 2.49 | 1.29 | 4.79 |
| Parasilo-4 | 0.50 | 2.51 | 1.51 | 4.95 | 1.01 | 2.39 | 3.23 | 4.85 |
| Parasilo-5 | 0.50 | 2.51 | 3.02 | 5.01 | 1.12 | 2.38 | 2.87 | 4.80 |
| Parasilo-6 | 1.51 | 2.52 | 1.01 | 4.96 | 1.94 | 2.44 | 4.81 | 4.76 |
| Parapide-1 | 1.01 | 28 kb | 3.03 | 28 kb | 1.94 | 28 kb | 3.23 | 28 kb |
| Parapide-2 | 2.02 | 28 kb | 3.02 | 28 kb | 1.29 | 28 kb | 1.87 | 28 kb |
| Parapide-3 | 1.51 | 28 kb | 2.52 | 28 kb | 2.41 | 28 kb | 5.35 | 28 kb |
| Parapide-4 | 2.02 | 28 kb | 1.51 | 28 kb | 0.70 | 28 kb | 1.94 | 28 kb |

*Table 6 Disk utilization of all nodes for different datasize*

Results of table 6 also show that our “Saksham” algorithm is successfully configured and all the blocks are rearranged to the nodes which have prirority-2 and disk utilization of nodes is also merely same. Figure 14 shows rearrangement time taken by “Saksham” algorithm.

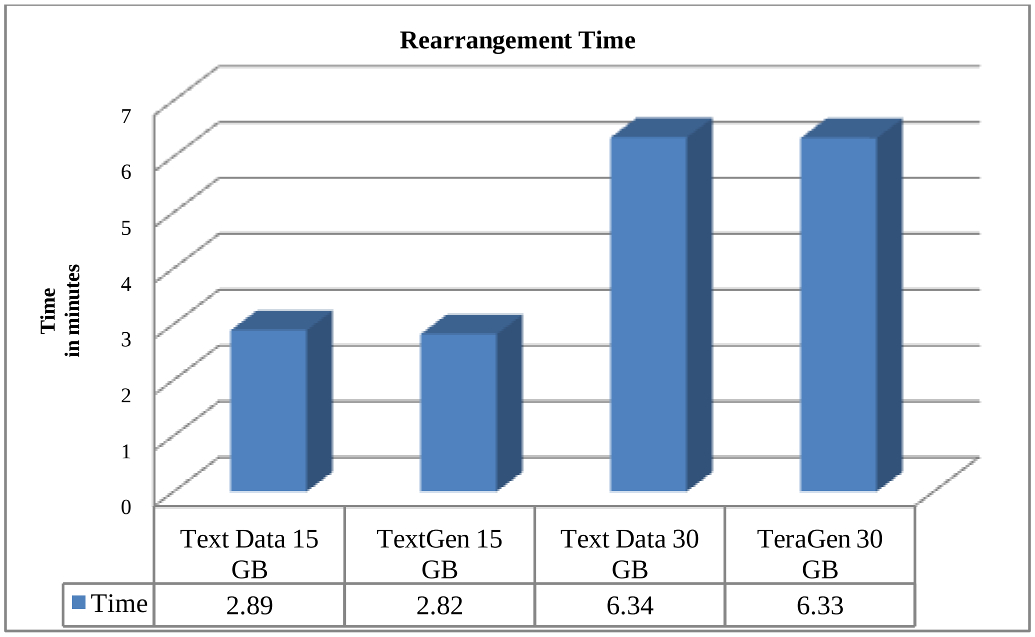


*Figure 14 Saksham block rearrangement time*

We successfully achieved control over block rearrangement based upon the priority assigned to the nodes. Next, we have assigned Node Labels to the nodes according to table 5 and using YARN resource manager we schedule the jobs according to given labels. This approach will prove the effectiveness of the proposed algorithm.

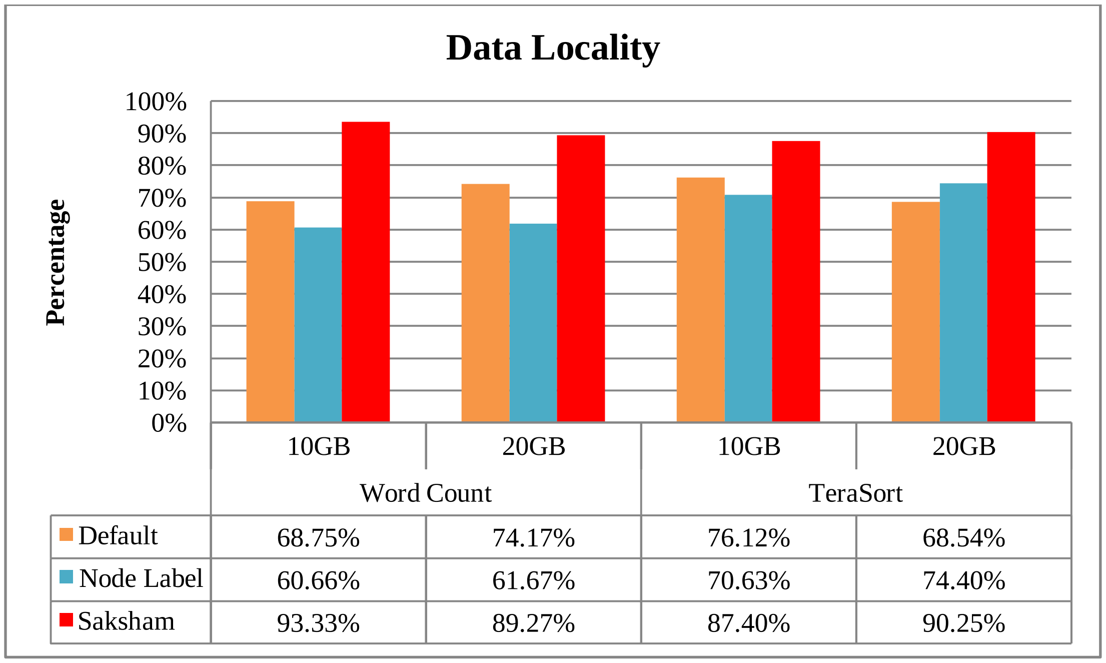
We have used two standard job applications for testing to prove the effectiveness of our proposed approach.

1. **WordCount:** Standard "bag of words"static test dataset is used for the counting job. Wordcount application counts the total no. of words from the file using MapReduce programming to achieve parallelism.
2. **TeraSort:** The TeraSort benchmark is the most well-known Hadoop benchmark for stress testing. To perform the sorting on data generated by TeraGen using MapReduce programming.

We use two datasets of size 10 GB and 20 GB for our experiment. We focus on two important parameters of performance improvement in Hadoop, data locality and job execution time. If data locality gets improve proportionally it improves MapReduce job processing time. We compared our strategy with default placement execution time and after applying only node labeling without "Saksham" balancing. Table 7 shows the result of total tasks launched, data local found and data locality in percentage. Figure 15 shows the comparative result of data locality achieved by various strategies. Results prove that our “Saksham” algorithm with node labeling approach achieves almost 90% data locality which is far better than other strategies.

| | | Default | | | Node Label | | | Saksham | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Jobs** | **Data Size** | **Total Task Launched** | **Data Local** | **Data Locality %** | **Total Task Launched** | **Data Local** | **Data Locality %** | **Total Task Launched** | **Data Local** | **Data Locality %** |
| **Word Count** | **10 GB** | 128 | 88 | 68.75% | 122 | 74 | 60.66% | 120 | 112 | 93.33% |
| | **20 GB** | 240 | 178 | 74.17% | 227 | 140 | 61.67% | 233 | 208 | 89.27% |
| **TeraSort** | **10 GB** | 134 | 102 | 76.12% | 126 | 89 | 70.63% | 127 | 111 | 87.40% |
| | **20 GB** | 267 | 183 | 68.54% | 250 | 186 | 74.40% | 236 | 213 | 90.25% |

*Table 7 Data locality results for various strategies*



| | Word Count 10GB | Word Count 20GB | TeraSort 10GB | TeraSort 20GB |
|---|---|---|---|---|
| 🟧 Default | 68.75% | 74.17% | 76.12% | 68.54% |
| 🟦 Node Label | 60.66% | 61.67% | 70.63% | 74.40% |
| 🟥 Saksham | 93.33% | 89.27% | 87.40% | 90.25% |

*Figure 15 Comparison of data locality achieved*

Last, we use default Hadoop schedulers for our test. We combine our “Saksham” algorithm plus node labeling and schedule the jobs for testing. We test using following schedulers to see the effectiveness: capacity and fair scheduler. Fair scheduler has three policies: Fair-FIFO, Fair-Fair and Fair-DRF. We compare the job execution time of our propose approach with default MapReduce execution, execution using node labeling w/o “Saksham” balancing. Figure [16-19] shows results of job execution time of two jobs (i.e. wordcount and terasort) using different schedulers.

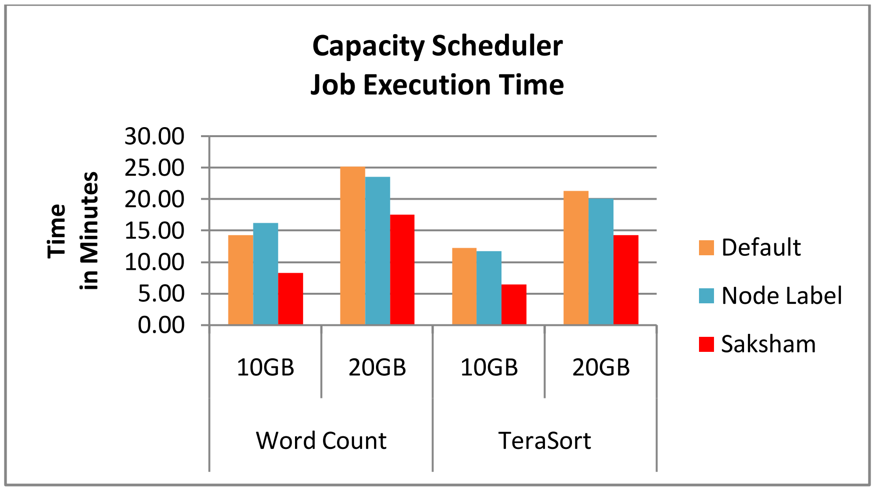


*Figure 16 Job Execution Time using Capacity Scheduler*

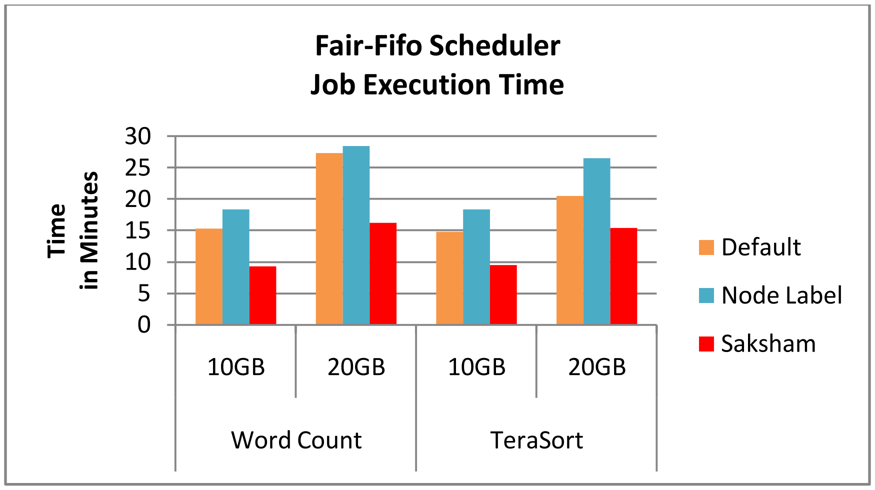


*Figure 17 Job Execution Time using Fair-FIFO Scheduler*

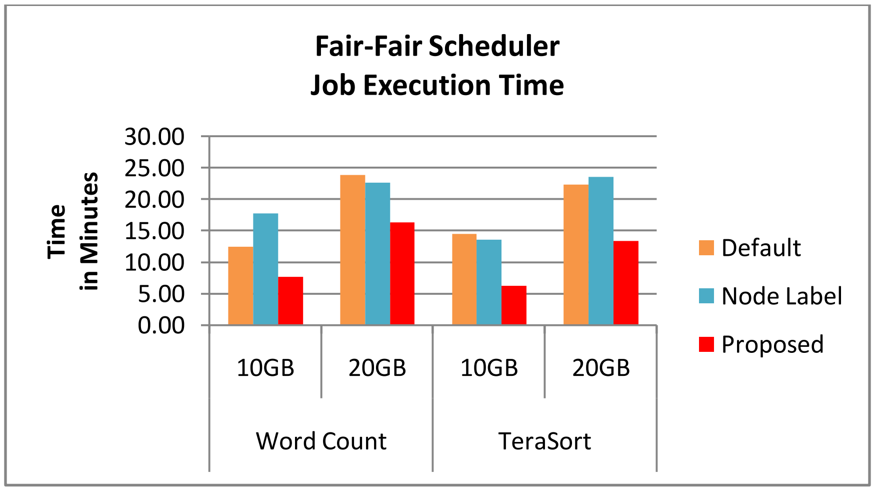


*Figure 18 Job Execution Time using Fair-Fair Scheduler*

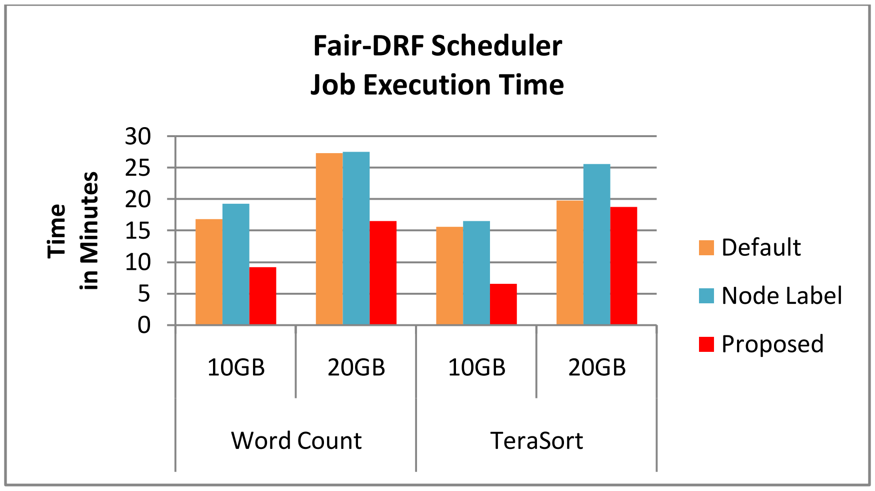


*Figure 19 Job Execution Time using Fair-DRF Scheduler*

Results show that mere implementation of node labeling creates overhead of the internode and interrack block transfer and increase the job execution time. But our "Saksahm" algorithm in combination with node labeling achieves optimized result. Below fig. 20 proves that Fair-Fair scheduling strategy works better compare to capacity, fair-fair, fair-drf policies.

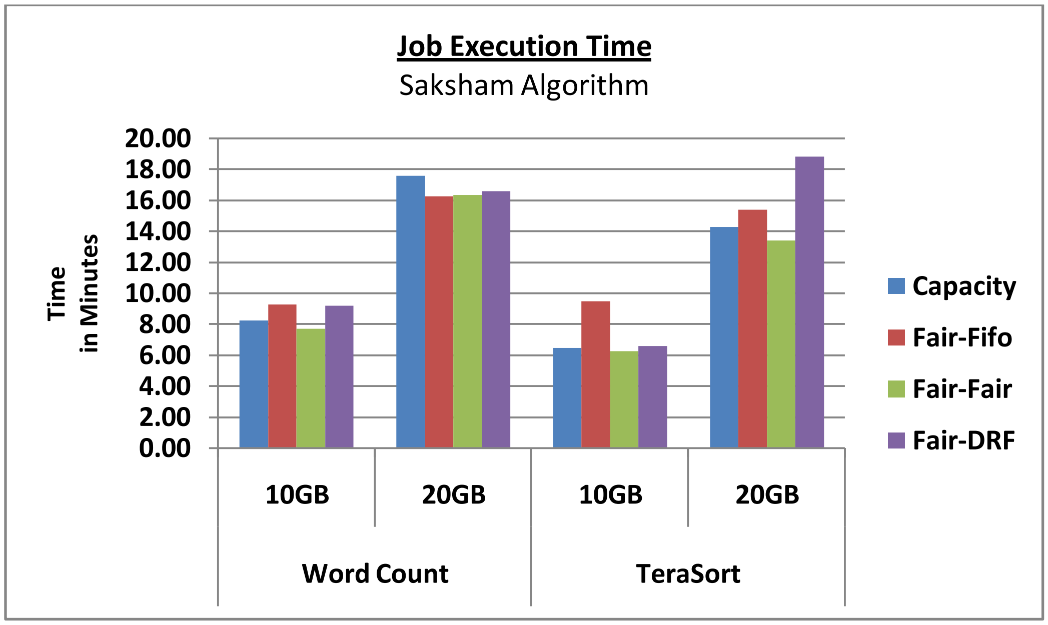


*Figure 20 Comparison of Job Execution Time using "Saksham" policy*

# Chapter 5: Conclusion

HDFS is an important core component of Apache Hadoop. Not only does it simply store data in a virtual file system, but HDFS also greatly affects and guides the MapReduce layer. By carefully rearranging blocks in the cluster, we can improve the performance of HDFS and thus also the overall performance of Apache Hadoop. This thesis contributes to the research of optimal data placement in large-scale server clusters.

Default block placement policy does not consider processing capability of nodes for placing data blocks. MapReduce job try to place process where data blocks are stored but it might be possible that the few nodes which are having data block may not have processing capability. Therefore, it is required to shift the process where processing capability is available and in that case, it may require to move blocks where the process is running. This affects to overall Hadoop performance which is an area of concern.

We propose Saksham: block rearrangement policy which leverages the processing capacity of CPU, during block placement. This approach will be helpful in MapReduce to minimize the internode and inter-rack transfer. We have demonstrated that we can place data blocks of a specific file to specific nodes only. This approach will not affect the overall load balancing of a cluster as rest of the files won't be affected. Experimental results prove that with the use of proposed scheme and Fair-Fair scheduler, Hadoop can achieve better performance for Big Data processing.

Considering all cluster nodes' processing capability plays an important role in a distributed computation framework. We show that distributing blocks and their replica rearrangement on desired cluster nodes, improves the performance of HDFS. The MapReduce layer is able to place a greater number of application copies onto cluster nodes with data locally available.

In conclusion, we believe that our "Saksham" block rearrangement algorithm is an improvement over the HDFS default blocks placement policy and "Saksham" policy combined with node labeling yield a greater performance over Hadoop default performance.

In future enhancement, it would be possible to replace default block placement policy with our "Saksham" policy for direct block placement. However, that would require a lot of modification in existing API. It is also possible to think over implementing dynamic priority assignment based on processing capability or hardware specifications.

## Research Paper Presented / Published

[1] Padole M., Shah A. (2018) Comparative Study of Scheduling Algorithms in Heterogeneous Distributed Computing Systems. In: Choudhary R., Mandal J., Bhattacharyya D. (eds) Advanced Computing and Communication Technologies. Advances in Intelligent Systems and Computing, vol 562. Springer, Singapore
Published [Scopus Indexed]

[2] Shah A., Padole M. (2018) Performance Analysis of Scheduling Algorithms in Apache Hadoop. Data Engineering and Applications Springer Book Series.
In process of Publication [EI-Compendex, DBLP, Scopus Indexed]

[3] Shah A., Padole M. (2018) Load Balancing through Block Rearrangement Policy for Hadoop Heterogeneous Cluster. ICACCI'18 Bangalore Conference.
In process of Publication - IEEE Xplore [Thomson Reuters Citation Index, Scopus Indexed]

[4] Shah A., Padole M. (2018) "Saksham: Resource Aware Block Rearrangement Algorithm for Load Balancing in Hadoop"
Paper submitted in International Journal

## References

[1] Andrew S. Tanenbaum, Maarten van Steen, (2002). Distributed Systems: Principles and Paradigms, Pearson Education Asia.

[2] SukumarGhosh, (2010). Distributed systems: an algorithmic approach. CRC press.

[3] George Coulouris, Jean Dollimore, Tim Kindberg, (2001). Distributed Systems: Concepts and Design, 4/E, Pearson Education Ltd.

[4] K. Hwang and F. A. Briggs, (1985). Computer Architecture and Parallel Processing. McGraw-Hill International Edition.

[5] Kshemkalyani, A.D. and Singhal, M., (2011). Distributed computing: principles, algorithms, and systems. Cambridge University Press.

[6] S. Srinivasan and N. K. Jha, (1999). "Safety and reliability driven task allocation indistributed systems." IEEE Transactions on Parallel and Distributed Systems, vol 10, no. 3, pp. 238-251.

[7] Kai Hwang, (1993). Advanced Computer Architecture: Parallelism. Scalability. Programmability McGraw-Hill International Editions.

[8] S.Bansal, P.Kumar and K.Singh,(2005) "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs." Journal of Parallel and Distributed Computing. Vol. 65, pp. 479-491.

[9] R. F. Freund and H. J. Siegel, (1993). "Introduction: Heterogeneous processing - guest editors introduction. IEEE Computer. vol. 26. no. 6. pp. 13-17.

[10] https://www.ibm.com/software/data/bigdata/what-is-big-data.html. [Accessed 10 Oct. 2018]

[11] Labrinidis A, Jagadish HV (2012) Challenges and opportunitieswith big data. Proc VLDB Endowment 5(12):2032–2033

[12] Chaudhuri S, Dayal U, Narasayya V (2011) An overview of business intelligence technology. Commun ACM 54(8):88–98

[13] Agrawal D, Bernstein P, Bertino E, Davidson S, DayalU,FranklinM, Gehrke J, Haas L, Halevy A, Han J et al (2012) Challengesand opportunities with big data. A community white paper developed by leading researches across the United States

[14] Chen, M., Mao, S. and Liu, Y., (2014). Big data: A survey. Mobile networks and applications, 19(2), pp.171-209.

[15] Hadoop, http://hadoop.apache.org [Accessed 10 Oct. 2018]

[16] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S. and Saha, B., (2013). Apache hadoop yarn: Yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (p. 5). ACM.

[17] Dean, J. and Ghemawat, S., (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp.107-113.

[18] Zheng, W.Sakellariou, R., (2013). Stochastic DAG scheduling using a Monte Carlo approach. Journal of Parallel and Distributed Computing. 73, 1673-1689.

[19] Munir, E., Mohsin, S., Hussain, A., Nisar, M., Ali, S., (2013). SDBATS: A Novel Algorithm for Task Scheduling in Heterogeneous Computing Systems. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. 43-53.

[20] Kwok, Y.Ahmad, I., (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys. 31, 406-471.

[21] Kanemitsu, H., Hanada, M., Nakazato, H., (2016). Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors. IEEE Transactions on Parallel and Distributed Sys-tems. 27, 3144-3157.
[22] Abdelkader, D.Omara, F., (2012). Dynamic task scheduling algorithm with load balancing for he-terogeneous computing system. Egyptian Informatics Journal. 13, 135-145.
[23] Wang, G., Wang, Y., Liu, H., Guo, H, (2016). HSIP: A Novel Task Scheduling Algorithm for He-terogeneous Computing. Scientific Programming. 2016, 1-11.
[24] Munir, E., Ahmad, S., Nisar, W., (2013). PEGA: A Performance Effective Genetic Algorithm for Task Scheduling in Heterogeneous Systems. In High Performance Computing and Com-munication & 2012 IEEE 9th International Conference on Embedded Software and Sys-tems (HPCC-ICESS), 2012 IEEE 14th International Conference. 1082-1087.
[25] Ahmad, S., Liew, C., Munir, E., Ang, T., Khan, S., (2016). A hybrid genetic algorithm for optimi-zation of scheduling workflow applications in heterogeneous computing systems. Journal of Parallel and Distributed Computing. 87, 80-90.
[26] Cardellini, V., Grassi, V., Presti, F., Nardelli, M., (2015). Distributed QoS-aware scheduling in storm. DEBS '15 Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems. 344-347.
[27] Arabnejad, H.Barbosa, J., (2014). List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. IEEE Transactions on Parallel and Distributed Systems. 25, 682-694.
[28] Khaldi, D., Jouvelot, P., Ancourt, C., (2015). Parallelizing with BDSC, a resource-constrained scheduling algorithm for shared and distributed memory systems. Parallel Computing. 41, 66-89.
[29] Li, K., Tang, X., Veeravalli, B., Li, K., (2015). Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems. IEEE Transactions on Computers. 64, 191-204.
[30] Barbosa, J. Moreira, B., (2011). Dynamic scheduling of a batch of parallel task jobs on heteroge-neous clusters. Parallel Computing. 37, 428-438.
[31] Choudhury, P., Chakrabarti, P., Kumar, R., (2012). Online Scheduling of Dynamic Task Graphs with Communication and Contention for Multiprocessors. IEEE Transactions on Parallel and Distributed Systems. 23, 126-133.
[32] Tang, Z., Jiang, L., Zhou, J., Li, K., Li, K., (2015). A self-adaptive scheduling algorithm for re-duce start time. Future Generation Computer Systems. 43-44, 51-60.
[33] Yuxiong, H., Liu, J., Hongyang, S., (2011). Scheduling Functionally Heterogeneous Systems with Utilization Balancing. IEEE International Parallel & Distributed Processing Symposium. 1187-1198.
[34] Hadoop Capacity Scheduler:https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html (Accessed 10 Oct. 2018)
[35] Hadoop Fair Scheduler:https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/FairScheduler.html (Accessed 10 Oct. 2018)
[36] Shah, A. and Padole, M., (2018). Load Balancing through Block Rearrangement Policy for Hadoop Heterogeneous Cluster. Paper Presented at the 7th International Conference on Advances in Computing, Communication and Informatics. 19-22 September 2018. Bangalore, India.
[37] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S. and Stoica, I., (2010). Delay scheduling: a simple technique for achieving locality and fairness in cluster

scheduling. In Proceedings of the 5th European conference on Computer systems (pp. 265-278). ACM.

[38] Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R.H. and Stoica, I., (2008). Improving MapReduce performance in heterogeneous environments. In Osdi (Vol. 8, No. 4, p. 7).

[39] Liu, Q., Cai, W., Shen, J., Fu, Z., Liu, X. and Linge, N., (2016). A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment. Security and Communication Networks, 9(17), pp.4002-4012.

[40] Liu, Y., Jing, W., Liu, Y., Lv, L., Qi, M. and Xiang, Y., (2017). A sliding window-based dynamic load balancing for heterogeneous Hadoop clusters. Concurrency and Computation: Practice and Experience, 29(3), p.e3763.

[41] Dharanipragada, J., Padala, S., Kammili, B. and Kumar, V., (2017), December. Tula: A disk latency aware balancing and block placement strategy for Hadoop. In Big Data (Big Data), 2017 IEEE International Conference on (pp. 2853-2858). IEEE.

[42] Anon, (2018). [online] Available at: https://github.com/fluxroot/hadaps. [Accessed 18 Oct. 2018].

[43] Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A. and Qin, X., 2010, April. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on (pp. 1-9). IEEE.

[44] Hsiao, H.C., Chung, H.Y., Shen, H. and Chao, Y.C., 2013. Load rebalancing for distributed file systems in clouds. IEEE transactions on parallel and distributed systems, 24(5), pp.951-962.

[45] Muthukkaruppan, K., Ranganathan, K. and Tang, L., Facebook Inc, (2016). Placement policy. U.S. Patent 9,268,808.

[46] Qureshi, F., Muhammad, N. and Shin, D.R., (2016). RDP: A storage-tier-aware Robust Data Placement strategy for Hadoop in a Cloud-based Heterogeneous Environment. KSII Transactions on Internet & Information Systems, 10(9).

[47] Qu, K., Meng, L. and Yang, Y., (2016), August. A dynamic replica strategy based on Markov model for hadoop distributed file system (HDFS). In Cloud Computing and Intelligence Systems (CCIS), 2016 4th International Conference on (pp. 337-342). IEEE.

[48] Meng, L., Zhao, W., Zhao, H. and Ding, Y., (2015). A Network Load Sensitive Block Placement Strategy of HDFS. KSII Transactions on Internet & Information Systems, 9(9).

[49] Dai, W., Ibrahim, I. and Bassiouni, M., (2017), June. An improved replica placement policy for Hadoop Distributed File System running on Cloud platforms. In Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on (pp. 270-275). IEEE.

[50] Fahmy, M.M., Elghandour, I. and Nagi, M., (2016), December. CoS-HDFS: co-locating geo-distributed spatial data in hadoop distributed file system. In Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (pp. 123-132). ACM.

[51] Park, D., Kang, K., Hong, J. and Cho, Y., (2016), April. An efficient Hadoop data replication method design for heterogeneous clusters. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 2182-2184). ACM.

[52] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F.B. and Babu, S., (2011), January. Starfish: a self-tuning system for big data analytics. In Cidr (Vol. 11, No. 2011, pp. 261-272).

[53] Hadoop.apache.org. (2018). Apache Hadoop 2.7.2 – YARN Node Labels. [online] Available at: https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/NodeLabel.html [Accessed 18 Oct. 2018].

[54] Team, D. (2018). Rack Awareness in Hadoop HDFS – An Introductory Guide – DataFlair. [online] Data-flair.training. Available at: https://data-flair.training/blogs/rack-awareness-hadoop-hdfs/ [Accessed 18 Oct. 2018].

[55] Grid5000.fr. (2018). Grid5000. [online] Available at: https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home [Accessed 18 Oct. 2018].