Chapter 4
Literature Work: Analysis & Comparison

In this section, literature study is divided into 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 Hadoop schedulers with their implementation and results are discussed. Last, alternative approaches adopted and designed by researchers and scientists to the implementation and improvement of load balancing algorithms and performance in Hadoop are discussed.

4.1 Study of Various Distributed File System

For Big Data storage there is a need of ideal Distributed File System (DFS) which can provide **a** wide range of support for distributed file storage, parallel file access and at the same time scalability for high performance. To achieve effective distributed computing infrastructure DFS is a core component of it. Therefore, it must support and fulfill the taxonomy of DFS described by *Thanh et al., 2008*. In the paper (Satyanarayanan, 1989) the author discussed various issues of DFS. The various distributed file systems have been surveyed in papers (Guan et al., 2000; Thanh et al., 2008; Depardon et al., 2013) by researchers and scientists.

As it was discussed in section 2 that idyllic DFS should be fault-tolerant, scalable and transparent, that apparently provides high availability and high performance. When Distributed file systems was not on its peak for distributed computing during 1980s-1990s, various DFS architectures such as NFS (Sandberg et al., 1985), Andrew file system (AFS) (Howard et al., 1988), CODA, (Satyanarayanan, 1990), Frangipani (Thekkath et al., 1997), and many more were introduced. In paper *Guanet al., 2000* has reviewed these and many other old days' well known

distributed file systems and found that none of those file systems could achieve all the characteristics of distributed computing.

Distributed computing has become an unquestionable choice in the age of 21st century when data and internet users are growing like anything. As data becomes Big Data, there a rises a need of distributed file system which can leverage over multiple storage devices, widely available and its performance does not get degraded. As no previous 1990s distributed file system met the requirement of Big Data storage and processing new file systems are devised.

Many distributed file systems are developed to overcome problems of old file systems discussed above. As our focus in thesis is on Big Data processing, here we have compared some well-known and most widely used distributed file system which fulfills the need of Big Data storage and process. We surveyed five distributed file systems, Google File System (GFS) (Ghemawat et al., 2003), Lustre (Braam and Zahir, 2002; Schwan, 2003), Ceph (Weil et al., 2006), Hadoop Distributed File System (HDFS) (Shvachko et al., 2010), and GlustreFS (Docs.gluster.org, 2018) which are application independent and used for general purpose. Here table 4.1 illustrates comparative study of various distributed file systems.

Google file system- also known as GoogleFS is designed by Google for distributed cluster storage. It is implemented by its own cluster storage platform for data-intensive applications. GFS uses commodity hardware machines named as chunk servers for distributed storage. GFS meets most of the design constraints of the distributed file system and efficiently utilizes low-cost machines.

Lustre is an open-source parallel file system (Lustre.org, 2018) that meets the requirement of High-Performance Computing file system. Lustre uses diskless workstation architecture rather than multiple storage units. Lustre maintains metadata on a shared server called Metadata Target (MDT) and also keeps the copy of those metadata for failover on Metadata Servers (MDS).

Ceph is a reliable, easy to manage, next-generation software based distributed object store that provides storage of unstructured data for Big Data

applications. Ceph is a Reliable, Autonomous, Distributed Object Store (RADOS) comprised of high availability, scalability and intelligent storage nodes.

	GFS	**Lustre**	**Ceph**	**HDFS**	**GlusterFS**
Storage Type	Object-based	Object-based	Object-based	Object-based	File-based
Metadata storage	Centralized	Centralized	Distributed	Centralized	Decentralized
Naming Service	Index	Index	CRUSH	Index	EHA
Operating System support	Linux Kernel	Linux Kernel	Linux, FreeBSD	Cross-platform	Linux, OS X, FreeBSD, NetBSD, OpenSolaris
Application	Stateful	Stateful	Stateful	Stateful	Stateful
Communic-ation	RPC, TCP/IP	InfiniBand	TCP	RPC, TCP/IP	TCP/IP, InfiniBand or Sockets Direct Protocol
Fault Tolerant	Yes	Yes	Yes	Yes	Yes
System availability	No failover	Failover	High	High	High
Data availability	Replication	No	Replication	Replication	RAID-like
Placement strategy	Auto	No	Auto	Auto	Manual
Replication	Asynchron ous	RAID-like	Synchrono us	Asynchron ous	Synchronou s
Load balancing	Auto	No	Manual	Auto	Manual
Security mechanism	No	Yes	Yes	No	No

* EHA – Elastic Hash Algorithm

Table 4.1 Comparison of Various DFS

HDFS is an open source distributed file system designed from GoogleFS. HDFS has achieved a great success compared to other file systems due to its adaptability to run on any hardware machines and also cross-platform support. HDFS maintains two namenode servers for providing high system availability.

GlustreFS is another open source software-based network file system for data storage which leverages on commodity hardware just like GFS and HDFS.

GlusterFS is specially designed for cloud storage and media streaming. Unlike other DFS, GusterFS uses hardware RAID for replica management.

Distributed file systems are standard solutions for cloud storage, cluster storage, and data centers. All five surveyed DFSs supports transparency and fault-tolerance. But there are no DFSs which fit all, means all provide distributed storage but all have their own differences lying in design and implementation. Therefore, the choice of DFSs is important depending upon the need of application and availability of resources. Furthermore, practical implementation and tests can justify the above theoretical comparison. But from the above comparison, we come down to a solution that for our research problem HDFS is a better approach as it supports heterogeneity of OS and open-source Hadoop framework support.

4.2 Study of Scheduling Algorithms in HeDC

Scheduling always remains an area of research in any computing technique. It is important to choose proper scheduling algorithms especially when you're dealing with heterogeneous environments. In heterogeneous environment resources are of distinct configuration. Consequently if proper resources are not allocated for execution of tasks, your task may get delayed and result in poor job scheduling.

For any scheduling primary objectives of algorithms are optimizing job completion time as well as resource utilization. Based on scheduling objectives they are categorized into two categories: Application Specific and System Specific. In case of Application Specific scheduling the main objective is to consider the performance of an application considering various parameters such as makespan, scheduling type and cost. While in case of system specific scheduling the areas of concern are resource utilization, resource multiplicity and load balancing to improve overall system performance. Figure 4.1 shows the objective-based classification of scheduling algorithms.

In application specific scheduling everything is monitored in terms of its impact on the application. For application specific scheduling it is very much important to consider heterogeneity hardware. For the performance essentiality the

measure set is widely distributed under time and money. Most of the distributed applications are concerned with time, like makespan, which is time taken to complete the job and overall economic cost of running application.

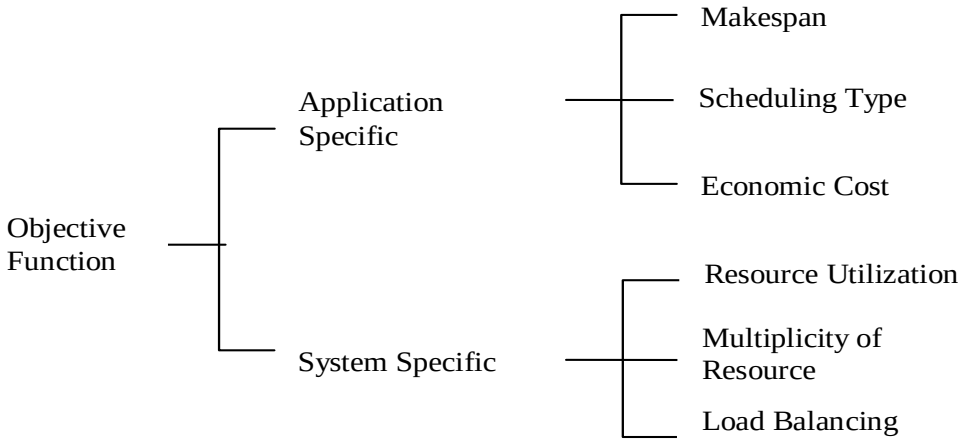


Figure 4.1 Objective-based Classification of Scheduling Algorithms

In System specific objectives are usually related to resource utilization, which resources are busy at what percentage of time. Types of resources and their effective utilization have a direct impact on the performance of an individual application and must be considered. System specific objectives are usually related to resource utilization, resource multiplicity and load balancing.

4.2.1 Comparative Study

The scheduling algorithms discussed below are widely adopted for scheduling on distributed environment. The table 4.2 shows comparative study of these algorithms highlighting scheduling approach used by a specific technique and probable enhancement to be considered for better performance.

Zheng and Sakellariou (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.

Munir et al. (2013) proposed standard deviation based algorithm for task scheduling (SDBATS) to reduce schedule length and speed up the scheduling by assigning task priority.

Kwok and Ahmad (1999) proposed to optimize the makespan by considering a wide range of techniques, genetic algorithm, randomized branch-and bound and graph theory. Authors have proposed many useful static, heuristic algorithms (e.g., HEFT, MCP, ETF, and DLS) but that will not be effective in today's era of big data.

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

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

He et al. (2011) proposed Multi-queue Balancing (MQB) algorithm that minimizes the makespan and maximizes the heterogeneous resource utilization.MQB has multiple queues for online scheduling to achieve better utilization and minimizing completion time.

Wang et al. (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 it identifies task priority and second it finds the best processor to execute the tasks.

Ahmad et al. (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. (2016) proposed hybrid genetic algorithm (HGA) is a hybrid combination of HEFT heuristic and PEGA genetic algorithm. It provides optimum makespan and load balancing over heterogeneous systems.

Cardellini et al. (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).

Arabnejad and Barbosa (2014) proposed Predict Early Finish Time (PEFT) to speed up and optimize makespan. It has two phases: task prioritizing and a processor selection which identifies task priority and allocates it to the best processor accordingly.

Khaldi et al. (2015) proposed static-heuristic scheduler called bounded dominant sequence clustering (BDSC) is an extension of DSC limiting memory constraints and the bounded number of processors. It is suitable for signal processing and image processing kind of application.

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

Barbosa and Moreira (2011) proposed parallel heterogeneous earliest finish time (P-HEFT) which is an extension of HEFT. P-HEFT supports parallel task DAG which provides optimized makespan that makes it suitable for image processing type of application.

Choudhury et al. (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 et al. (2015) proposed self-adaptive reduce scheduling (SARS) for Hadoop platform. During MapReduce phase, it reduces waiting time by selecting adaptive time to schedule the reduce task. This method reduces turnaround time.

Table 4.2 Comparison of Various Scheduling Algorithms

Sr. No	Algorithm	Key Parameter	Scheduling Type	Nature of Task	Environment	Objective	Comments	Algorithm compared with
1	Monte Carlo based DAG* scheduling approach	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 Algorithm for Task Scheduling - SDBATS	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* [10]	Makespan, Degree of multiprogramming	Global, Static, Sub-optimal, Heuristic	Simultaneous Tasks	Bounded Number of Homogeneous Processors	To minimize make span	Standard algorithms for static task scheduling.	-
4	Clustering for Minimizing the Worst Schedule Length - CMWSL	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 Algorithm

Sr. No	Algorithm	Key Parameter	Scheduling Type	Nature of Task	Environment	Objective	Comments	Algorithm compared with
5	Cluster-ing Based HEFT with Dupli-cation – CBHD	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	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	Schedule length ration, efficiency	Global, Static	Simultaneo us Task	Heterogeneou s 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	Optimal mapping, sequence of execution	Global, Static Optimal	Random Task	Heterogeneou s 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 with
9	Hybrid Genetic Algorithm – HGA	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	Latency, availability, system utilization	Global, Dynamic, Adaptive Scheduler	Distributed Stream Processing	Peersim Simulator	Improve performance by adaptive scheduler for distributed	Storm based scheduler for overall system performance and quality of services.	cRR*, cOpt*
11	Predict Earliest Finish Time – PEFT	Efficiency, Makespan, better schedule length ratio	Global, Static	Workflow	Not mentioned	Forecast the cost table and schedule the task accordingly to achieve low complexity	Suitable when all tasks are known along with their complexity and dependency.	Look-ahead, HEFT,HCPT*, PETS*, HPS*
12	BDSC Hierarchical BDSC - HBDS	Resource Management, Speedup, Bounded Number of Processors	Global, Static, Sub-Optimal, Heuristic	Parallel Task Execution	PIPS Platform	Effective resource management and speedups on shared and distributed environment	Suitable for signal processing, image processing application. For BNP system is robust	HLFET, ISH, MCP, HEFT

Sr. No	Algorithm	Key Parameter	Scheduling Type	Nature of Task	Environment	Objective	Comments	Algorithm compared with
13	Stochastic Dy-namic Level Sched-uling - (SDLS)	Makespan, Speedup, and makespan standard deviation	Global, Dynamic, Distributed	Precedenc e constraine d 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	Makespan, Heterogeneity, Parallel task execution	Global, Dynamic	Simultane ous Tasks	Not mentioned	To minimize the completion time by parallel execution.	Suitable for image processing application. Best for multiple-mixed jobs.	Heterogeneo us Parallel Task Scheduler (HPTS)
15	Online Scheduling of Dynamic Task Graphs	Number of Processors, Memory	Global, Dynamic	Real time Tasks	Not mentioned	To provide runtime scheduling	Interprocessor communications of fixed number of homogeneous processors.	Multiprocess or scheduling algorithm
16	Self Adaptive Reduce Scheduling – SARS	Reduce completion time	Global, Dynamic, Distributed, Optimal	Batch processin g	Hadoop Map-Reduce Cluster	To minimize average completion time and response time	Suitable to big data application processing.	FIFO, Fair , Capacity 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

4.3 Study of Scheduling Algorithms in 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 the Apache Hadoop framework. Mere implementation of the application on Distributed Systems may not make optimal use of available resources. Hence, optimizing scheduling algorithms may further improvise the use of resources. In this study, we implement and test the results of the scheduling algorithms discussed in next sub section. Later we discuss how fine-tuning of scheduling policies can be used to achieve better performance of different applications, which have been implemented and tested in Apache Hadoop. The results represented here have been published in (Shah and Padole, 2018).

4.3.1 Experimental Environment, Workload, Performance Measure & Queue Configuration

In this paper, we evaluate the performance of two Hadoop schedulers by using three built-in scheduling policies (i.e., FIFO, Fair, and DRF) and test it in context to different queue settings. The performance metrics include six dimensions, which are data locality, latency, completion time, turnaround time, CPU and memory utilization. For our experiment, we implemented Hadoop-2.7.2 cluster (Refer to Appendix I). The cluster consisted of 1 master node and 11 slave nodes. The important information of the cluster configuration is shown in table 4.3.

Six big-data jobs were chosen for this experiment. Those were WordCount, WordMean, TeraSort, and PiEstimator. Here WordCount and WordMean were CPU-intensive jobs while TeraSort and PiEstimator were more memory intensive job.

WordCount calculates total number of words in a file while WordMean calculates average length of words in a given file. TeraSort performed on data generated by TeraGen and PiEstimator estimates the value of Pi. Our workload is a combination of CPU and memory intensive jobs to check the performance of Hadoop schedulers in an experimental environment in a very effective way. Details of each job are given in table 4.4.

Configured Parameter	Master Node	Cluster Nodes
Nodes (In Cluster)	1 (NameNode)	11 (DataNode)
Network	1 Gbps	1 Gbps
CPU	Pentium Dual-Core CPU @ 3.06 GHZ * 2	Pentium Dual-Core CPU @ 3.06 GHZ * 2
Cache	L1- 64 KBL2- 2 MB	L1- 64 KBL2- 2 MB
RAM	4 GB	22 GB (11 Nodes * 2 GB per node)
Disk	500 GB SATA	500 GB SATA
Block size	128 MB	128 MB
CPU Cores	2	22 (11 Nodes)
OS	Ubuntu 14.04 LTS	Ubuntu 14.04 LTS

Table 4.3 Hadoop Cluster Configuration

Job	# Maps	# Reducers	Job Size
WordCount	24	10	2 GB
WordMean	24	1	2 GB
TeraSort-1	15	10	2 GB
TeraSort-2	25	10	3 GB
Pi-1	20	1	1,00,000 Samples
Pi-2	50	1	1,00,000 Samples

Table 4.4 Total Workload Distribution

The performance of the Hadoop schedulers has been evaluated by considering following parameters:

- **Locality:** HDFS maintains the copy of the data splits across the datanodes. When any job is executed, MapReduce divides the job among multiple tasks, which is submitted for execution on multiple datanodes. Each mapper

requires the copy of data to process, if the data is not available on the same datanode it will find and bring a copy of data from other datanodes over the network, to the node where it is required. So if the data is available on the same node, it is called data local, if it is available on the same rack, it is called rack local and if both scenarios fail then it will be copied from a different rack. So if a job finds most of the data locally then completion time of job would be better than rack local or different rack data. (The data found on another node but within the same cluster is known as data on same rack. If data is residing on a datanode in a different cluster, it is referred as ~~a~~ different rack)

- **Latency:** It is the time that a job has to wait, until getting scheduled, after the job is submitted.
- **Completion Time:** It is the difference of finish time and start time of a job. It is the sum of actual execution time of the job and the waiting time, if any.
- **Turnaround Time:** It is the total amount of time elapsed between submission of the first job and till the completion of the last job, in the queue.
- **CPU & memory utilization:** Hadoop counters provide the time spent by the job on CPU and total memory bytes utilized by the job.

For the experiment purpose three types of job queues have been configured: a.) Single Queue b.) Multi-Queue c.) Mixed-Multi Queue

Single Queue

In a single queue, all the resources of a cluster will be used by one queue only. All jobs will be entered and scheduled according to the scheduling policy and availability of resources. Capacity Scheduler will be configured with FIFO while Fair Scheduler can be configured with FIFO, FAIR or DRF policy. These four schemes (Cap-FIFO, Fair-FIFO, Fair-FAIR, and Fair-DRF) will be evaluated based on discussed six variables using workload listed in table 4.5.

Multi-Queue

Here for our experiment, we have considered three queues where each queue will be running a similar kind of application. Three queues named "A", "B" and "C" is configured with 30%, 40% and 30% resources respectively of the total resources available. These queues have been kept soft so that each queue can use 2 times of its configured capacity for elasticity purpose. Jobs are allocated to the queues as per given table 4.5.

Mixed-Multi Queue

In this case, to test the performance variation we alter jobs to different queues. If we put different types of applications in the same queue, performance is measured as to how it affects the performance in such situation. For our experiment mixed-jobs are allocated to the queues as per table 4.5.

4.3.2 Performance Evaluation

For our experiment, each application has been executed 5 times to validate the results of the evaluated performance. Performance evaluation is carried out without changing its default settings except for the queue settings of the schedulers.

Multi-Queue		Mixed-Multi Queue	
Queue	Jobs	Queue	Jobs
A	WordCount	A	WordCount
A	WordMean	A	TeraSort-1
B	TeraSort-1	B	Pi-1
B	TeraSort-2	B	TeraSort-2
C	Pi-1	C	WordMean
C	Pi-2	C	Pi-2

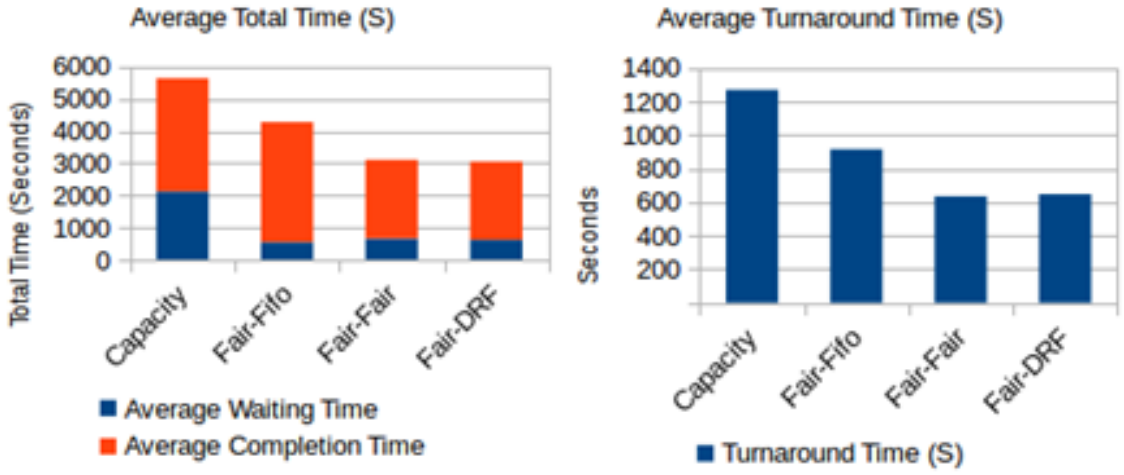
Table 4.5 Workload Assignment to the Queue

Single Queue

All six big-data applications are entered into a single queue with normal 2 seconds of delay. By looking at the results in fig. 4.2 (a) it is quite evident that in Cap-FIFO scheduler job waiting time and completion time is comparatively quite high as compared to other three scheduling policies. Fair-FIFO has less waiting time as a job

gets scheduled by a fair policy but completion time is more compared to Fair-Fair and Fair-DRF policy. While Fair-Fair and Fair-DRF perform better, it has almost similar job waiting time and completion time. As shown in fig. 4.2 (b) in terms of turnaround time also Fair-Fair and Fair-DRF outperforms other scheduling policy.

As shown in table 4.6 in terms of data locality, 74.40% tasks are data local in Fair-DRF which means that Fair-DRF finds maximum data task as data local out of the total tasks launched. Moreover, Fair-DRF provides more resource efficiency as total task launched by Fair-DRF is less compared to others. In terms of CPU time & memory usage shown in fig. 4.3, Fair-Fair is better as it utilizes less CPU time and uses the physical memory effectively.



(A) (B)

Figure 4.2 (A) Average Total Time (B) Average Turnaround Time (Single Queue)

	Capacity	Fair-FIFO	Fair-Fair	Fair-DRF
Single Queue Locality				
Average Total_Launched	201	179	176	168
Average Data_Local	126	112	114	125
Percentage	62.69%	62.57%	64.77%	74.40%
Multi-Queue Locality				
Average Total_Launched	211	178	169	172
Average Data_Local	93	121	129	119
Percentage	44.08%	67.98%	76.33%	69.19%
Mixed-Multi Queue Locality				
Average Total_Launched	196	173	169	166
Average Data_Local	85	122	122	120
Percentage	43.37%	70.52%	72.19%	72.29%

Table 4.6 Data Locality
(Percentage shows how many % of data found data local out of the total task launched)

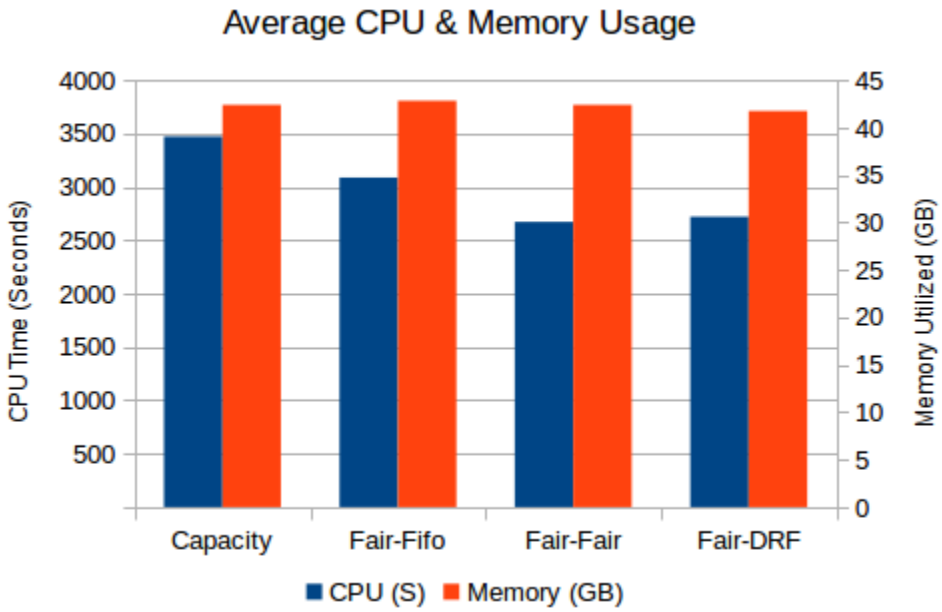


Figure 4.3 CPU & Memory Usage (Single Queue)

Multi-Queue

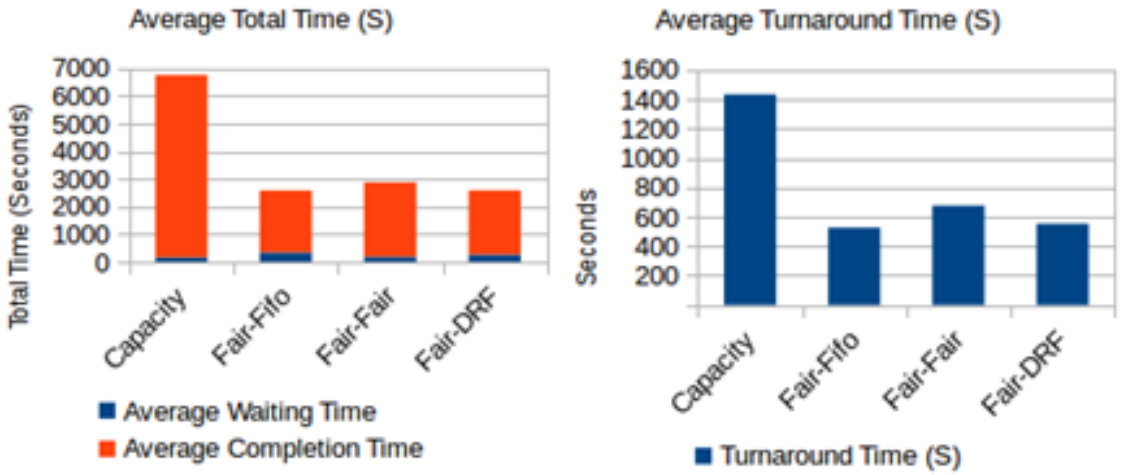
As illustrated by fig.4.4 (a), in Cap-FIFO scheduler job waiting time is less but completion time is comparatively quite higher than other three scheduling policies. Fair-FIFO performs better in terms of the job waiting time and completion time in comparison with Fair-Fair and Fair-DRF policy. Moreover, fig. 4.4 (b) depicts that Fair-FIFO is more efficient than Fair-Fair and Fair-DRF scheduling policy in terms of turnaround.

In terms of data locality, as illustrated by table 4.6, 76.33% tasks are data local in Fair-Fair indicating that Fair-Fair finds maximum data task as data local out of the total tasks launched. Furthermore, Fair-Fair supplements more resource efficiency as total task launched by Fair-Fair is less compared to others. In terms of CPU time and memory usage shown in fig. 4.5, Fair-FIFO is better since it utilizes less CPU time and uses the physical memory constructively.

Mixed-Multi Queue

It is quite obvious from fig.4.6 (a) that in Cap-FIFO scheduler job waiting time is less but completion time is considerably very high as compared to other three scheduling policies. Fair-Fair and Fair-DRF perform quite alike in terms of the job

waiting time but if total time is considered, then Fair-DRF performs better than rest of all scheduling policies. Fig. 4.6 (b) suggests that Fair-DRF is more effective than Fair-Fair and Fair-DRF scheduling policy in terms of turnaround.



(A) (B)

Figure 4.4 (A) Average Total Time (B) Average Turnaround Time (Multi-Queue)

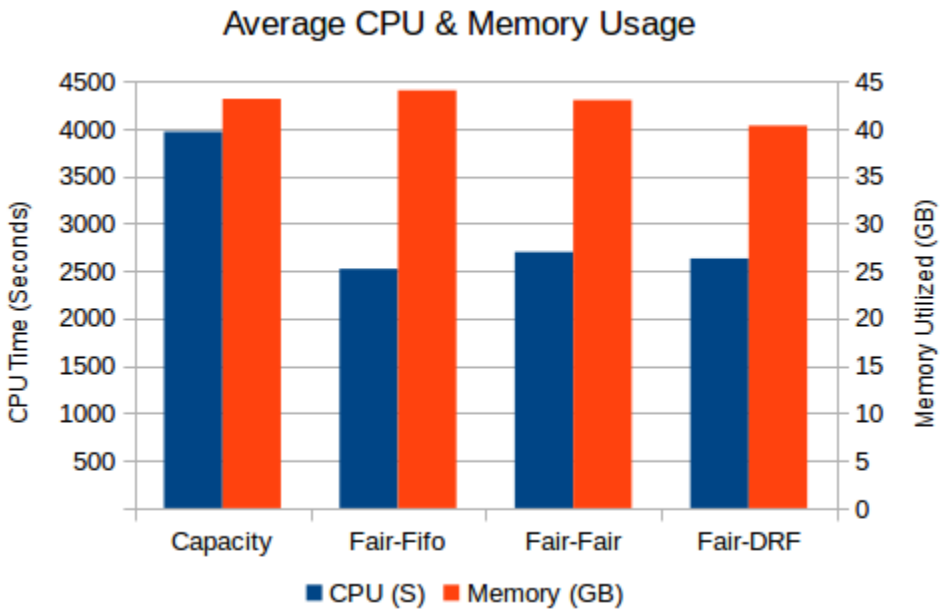
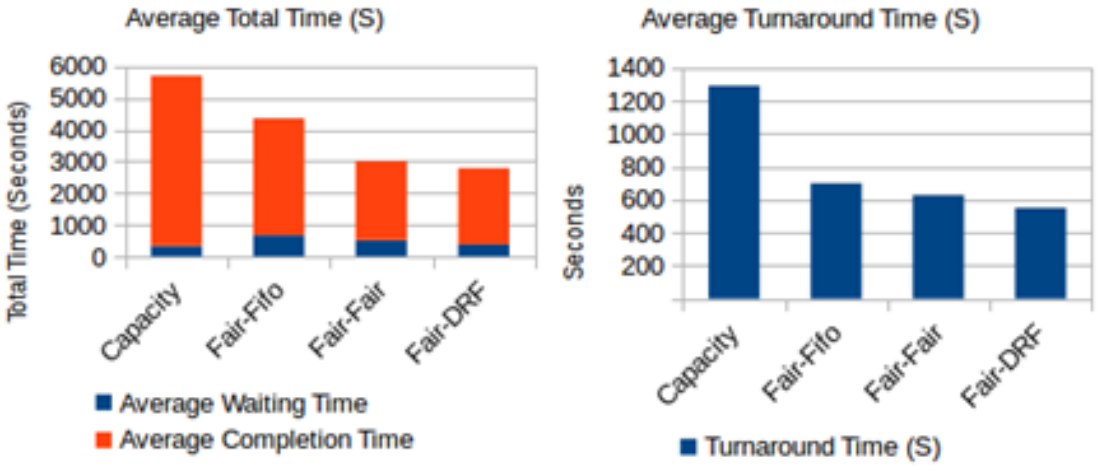


Figure 4.5 CPU & Memory Usage (Multi-Queue)

Table 4.6 elucidates that in terms of data locality, 72.29 % and 72.19 % tasks are data local in Fair-DRF and Fair-Fair respectively. It indicates that Fair-Fair and Fair-DRF find maximum data task as data local out of the total tasks launched. Moreover, Fair-DRF gives more resource efficiency as total tasks launched by Fair-DRF is less compared to others. In terms of CPU time and memory usage shown in fig. 4.7, Fair-Fair & Fair-DRF performs similarly.



(A) (B)

Figure 4.6 (A) Average Total Time (B) Average Turnaround Time (Mixed-Multi Queue)

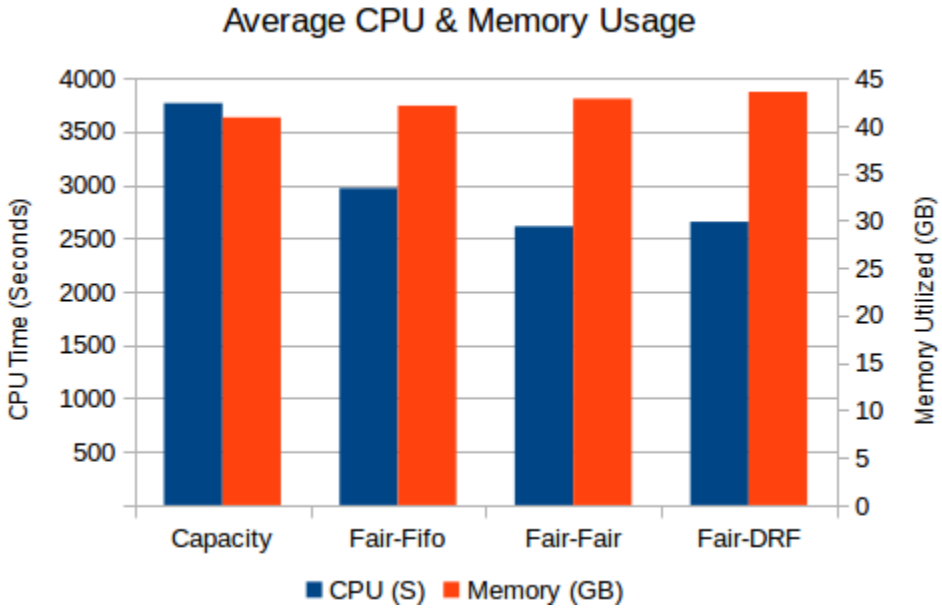


Figure 4.7 CPU & Memory Usage (Mixed-Multi Queue)

Concluding Remarks

In this experiment, various Hadoop scheduling algorithms have been evaluated based upon latency time, completion time and data locality. For the experiment purpose, six big data applications have been implemented using three different scheduling queue configurations such as Single-Queue, Multiple-Queue and Mix-Multiple Queue. Various experiments were conducted by fine-tuning scheduling policy for Hadoop environment. The results of the experiments are presented here which may be useful in selecting a scheduler and scheduling policy depending upon application that one wants to run. Based on the results, following conclusions can be drawn:

1. In single queue, Fair-DRF outperforms with reference to execution time and effective resource usage capacity as compared to other three. Its only flaw is that CPU usage time is a bit higher than Fair-Fair scheduler.
2. In multi-queue, Fair-FIFO is the best option if we consider workload waiting time, completion time, turnaround time and CPU usage. Fair-Fair is better only when resource utilization is important.
3. In mixed-multi queue, Fair-DRF is the most appropriate choice with respect to resource utilization and workload execution performance.

The results which are drawn here can be application dependent and future researcher can test the same with different application types. The future enhancements can include testing the impact of delay scheduler on capacity and fair scheduler. It is presumed that delay scheduling may make a significant impact on the performance of the scheduler.

4.4 Study of Hadoop Performance Improvement Techniques

To improve the performance of Hadoop many researchers have worked with diverse approaches. In distributed computing, load balancing is the key area which affects overall performance significantly, since the system may consist of thousands of computers in the cluster. Many researchers have worked on performance improvement through effective load balancing using various custom-designed algorithms and programming models. In paper authors (Shah and Padole, 2018), have summarized notable research contribution for load balancing by scheduling (Zaharia et al., 2008; 2010), load balancing during job processing (Liu et al., 2016; 2017) and load balancing using custom block placement (Dharanipragada et al., 2017; Anon, 2018; Xie et al., 2010; Hsiao, 2013). In this section, we have summarized some of the noteworthy work done to achieve better performance in Hadoop using load balancing and custom block placement strategy.

In the paper (Muthukkaruppan et al., 2016) authors proposed the approach which places the blocks based on a region placement policy. Data is stored into the plurality of regions rather than the plurality of nodes. Therefore, the complete

replica of the region can be stored into a contiguous portion of data. This policy achieves fault tolerance and data locality for region based cluster storage.

Authors of paper (Qureshi et al., 2016) presented 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. The experiment proves that it reduces imbalance of the cluster.

In the paper (Qu et al., 2016) authors come up with a dynamic replica placement which works on Markov probability model and places replica homogeneously across the racks. Results show better job completion time compared to HDFS and CDRM apart from a uniform distribution of replicas across all the nodes.

In the paper (Meng et al., 2015) authors proposed a 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 relevance to throughput and storage space utilization.

In the paper (Dai et al., 2017) authors proposed improved slot replica placement policy which considers the heterogeneity of nodes and partitions of 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 the paper (Fahmy et al., 2016) authors presented a strategy which tracks spatial characteristics of data to co-locate them. If data blocks are geographically distributed across multiple data centers irrespective of the location where a job runs, 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 (Park et al., 2016) authors proposed probability based DLMT (Data Local Map Task Slot) approach which adjusts data placement rate along with replica eviction policy to improve Hadoop performance and cluster space utilization respectively.

In paper (Herodotou et al., 2011) authors presented a model called "Starfish" which dynamically adjusts the Hadoop parameter according to the workload of the job. Starfish works with each phase of Hadoop, starting with job level tuning, real-time parameter adjustment and finishing with process scheduling. It achieves great performance compared to default Hadoop set up, placement policy and scheduling scheme. Table 4.7 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 stores one complete replica of the region on a 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 a node for balancing.	Authors have successfully demonstrated 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 an important justification.

Network sensitive strategy	Strong fault tolerant 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 imbalance 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	Reduces total execution time and network bandwidth.	Proposed algorithm which is aware of geospatial 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 have 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 4.7 Various Hadoop Performance Improvement Techniques