Thermal Aware Scheduling in Hadoop MapReduce Framework

by

Sayan Kole

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2013 by the
Graduate Supervisory Committee:

Sandeep Gupta, Chair
Dijiang Huang
Georgios Varsamopoulos

ARIZONA STATE UNIVERSITY

December 2013

ABSTRACT

The energy consumption of data centers is increasing steadily along with the associated power-density. Approximately half of such energy consumption is attributed to the cooling energy, as a result of which reducing cooling energy along with reducing servers energy consumption in data centers is becoming imperative so as to achieve greening of the data centers. This thesis deals with cooling energy management in data centers running data-processing frameworks. In particular, we propose thermal aware scheduling for MapReduce framework and its Hadoop implementation to reduce cooling energy in data centers. Data-processing frameworks run many low-priority batch processing jobs, such as background log analysis, that do not have strict completion time requirements; they can be delayed by a bounded amount of time. Cooling energy savings are possible by being able to temporally spread the workload, and assign it to the computing equipments which reduce the heat recirculation in data center room and therefore the load on the cooling systems. We implement our scheme in Hadoop and performs some experiments using both CPU-intensive and I/O-intensive workload benchmarks in order to evaluate the efficiency of our scheme. The evaluation results highlight that our thermal aware scheduling reduces hot-spots and makes uniform temperature distribution within the data center possible. Summarizing the contribution, we incorporated thermal awareness in Hadoop MapReduce framework by enhancing the native scheduler to make it thermally aware, compare the Thermal Aware Scheduler(TAS) with the Hadoop scheduler (FCFS) by running PageRank and TeraSort benchmarks in the BlueTool data center of Impact lab and show that there is reduction in peak temperature and decrease in cooling power using TAS over FCFS scheduler.

ACKNOWLEDGEMENTS

I would like to thank Dr Sandeep Gupta and Dr Georgios Varsamopoulos for their guidance and motivation during the course of my study at Arizona State University. I'm thankful to my committee member Dr Dijiang Huang for agreeing to being a part of the committee and giving valuable feedback. I would also like to thank Zahra Abbasi for her help, guidance as well as timely suggestions.

I thank the National Science Foundation through award #0855277 for funding this research. Lastly I would like to thank my parents for being a source of constant source of support and inspiration.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

LIST OF TABLES

Chapter 1

INTRODUCTION

The energy consumption of data centers is increasing steadily along with the associated power-density. Approximately half of such energy consumption is attributed to the cooling energy, as a result of which reducing cooling energy along with reducing servers energy consumption in data centers is becoming imperative so as to achieve greening of the data centers. The Green Grid, a consortium of data center technology companies, is devoted to increasing public awareness on energy and thermal efficiency of data centers; to that purpose it has proposed metrics on energy efficiency of data centers, e.g. Belady *et al.* (2008). Although resource and workload management is imperative to achieve good energy efficiency, many popular workload management frameworks are not thermal-aware as shown in Table 1.1, particularly, Hadoop, White (2012), being the focus of this research. In data centers with limited power budget, incorporating thermal awareness in the task distribution framework is a cost-effective way to operate the data center. The main energy consumed by the data center is in the form of two components, the cooling power and the computing power. The power usage effectiveness (PUE) of a data center, which is defined as the total power over the computing power is affected by the computing and the cooling power, Belady *et al.* (2008). A large PUE is a strong indication of large cooling power, since the cooling system is the biggest consumer of the non-computing power in a data center (followed by power conversion and other losses). Thermal aware resource management is of utmost important for such data centers to improve the cooling energy efficiency. According to US Department of Energy[1], average data centers PUE is around 1.7,

[1]http://www1.eere.energy.gov/manufacturing/datacenters/about.html

Table 1.1: Improving energy efficiency in Data Centers

Design	Equipment	Software Framework
Cold aisle, hot aisle arrangement; aisle isolation	Low power states and DVFS (e.g. Intel's SpeedStep, AMD's PowerNow); Low power electronics (Intel Atom)	Popular commercial frameworks are not thermal-aware e.g. Hadoop, Rocks, Moab are not thermal-aware

which means that $0.7/1.7 \sim 41\%$ of the power is consumed in cooling the data center. Hadoop, being popularly used by enterprises that usually have average data centers, incorporating thermal awareness into it would benefit them.

In this research we propose to incorporate thermal awareness in Hadoop MapReduce framework so as to contribute in reducing cooling power in the data centers it runs. The proposed scheme classifies servers as "HOT","COLD" and "MEDIUM" based on its initial CPU temperature. We then modify the Hadoop scheduler to allocate more tasks to the "COLD" servers by making the "HOT" and "MEDIUM" servers wait for an interval of time. Finally we evaluate and compare the Thermal Aware Scheduler(TAS) with the native Hadoop scheduler (FCFS) by running CPU intensive and I/O intensive benchmarks on BlueTool Infrastructure of Impact Lab and show that TAS decreases peak and average temperatures from those in FCFS and also reduces cooling power. The novelty of the work is incorporating thermal awareness in Hadoop by preferential distribution of tasks to thermal efficient servers and showing that CPU intensive tasks gain most in terms of temperature reduction if Hadoop is

made thermally aware.

The cooling energy depends on two factors 1) the cooling demand driven by the power distribution and the redline temperature and 2) the cooling behavior i.e. the behavior of the Computer Room Air Conditioner to meet the cooling demand. Of particular concern is the recirculation and intermixing of the hot air generated from the servers running the jobs with the cold air supplied from the CRAC. The heat recirculation depends on the data center layout and can cause hot-spots which in turn increase the cooling demand.

In past research, Abbasi *et al.* (2012), has proposed to reduce the hot-spots using a thermal-aware workload/job management scheme. Such a scheme is aware of the heat recirculation caused by the servers and the Computer Room Air Conditioner (CRAC) and assigns tasks intelligently to servers so as to minimize the heat re-circulation and thus prevent hot-spots, thereby decreasing the cooling energy. Banerjee *et al.* (2010) have proposed coordinated cooling aware job placement and cooling management algorithm, Highest Thermostat Setting that takes into account the dynamic behavior of the units and places jobs in a way that reduces cooling demand for the CRAC. Varsamopoulos *et al.* (2009) have shown the importance of using realistic cooling models in analyzing cooling energy of thermal aware job scheduling algorithms.

The demand for Big Data analytics processing in data centers is growing rapidly, for which Apache Hadoop is the most popular service infrastructure White (2012). Apache Hadoop is an open source implementation for parallelizing the processing of web based data. It comprises of a framework called MapReduce, with its supporting file system named as Hadoop Distributed File System. A Hadoop infrastructure contains master/slave architecture with a master node allocating tasks to the slave nodes on a first come first serve basis without incorporating any thermal awareness while performing such an allocation. Despite popularity, the current Hadoop has a simple scheduling framework, lacking energy management options and in particular cooling energy management. Therefore, we propose a thermal aware scheduling algorithm for Hadoop, implement and test it using realistic traces and thereby demonstrate that such a cooling technique incorporated in Hadoop scheduler achieves reduction in cooling energy

Thermal aware workload scheduling is extensively studied for batch jobs of HPC data centers, as well as Internet data centers so as to optimize cooling energy. Despite popularity for being service infrastructure of big data analysis in data centers, Hadoop has not been studied for cooling energy optimization. There are very few studies which discuss energy management solutions in Hadoop, but to our knowledge thermal aware scheduling has not been developed/evaluated yet. Particularly,Goiri *et al.* (2012), have proposed GreenHadoop which schedules the MapReduce jobs so as to maximize the green energy consumption within the time bounds for the job. Also Li *et al.* (2011) proposed a technique to take into account the power profile of the data center in order to optimize the throughput of MapReduce. The result of the above work motivate improving Hadoop scheduling framework to optimize energy consumption and throughput. Given that the efficiency of thermal aware scheduling has been verified in various data centers with different applications, we propose to enhance Hadoop scheduling to incorporate the thermal awareness so as to optimize cooling energy.

The contribution of this thesis is as follows 1.Coming up with a classification scheme for servers based on their thermal efficiency which is light weight and fast 2.Modified Hadoop Scheduler which distributes tasks based on requests(“pull”) to a thermal aware scheme of distributing tasks to thermal efficent servers (”push”) 3.Investigated the tradeoff between thermal awareness and the delay interval so as to meet job deadline requirement 4.Evaluation results show that CPU intensive jobs benefit more from thermal awareness than I/O intensive jobs

Chapter 2 introduces the Apache Hadoop framework and its components. In Chapter 3 we describe the current schedulers available in Hadoop and what their features are. Chapter 4 presents our description of data center layout as well as thermal awareness in data center. We then describe the Thermal Aware Schedulers in literature and explain in detail our proposed Thermal Aware Scheduler (TAS). Chapter 5 evaluates and compares TAS with FCFS by running them on CPU and I/O intensive benchmarks and shows that TAS provides reduction in peak and average temperature as well As cooling power over FCFS. Conclusion and Future work are presented in Chapter 6.

Chapter 2

APACHE HADOOP FRAMEWORK

In this chapter we describe the Apache Hadoop MapReduce Framework briefly focusing on the components relevant for our research. We talk about the Hadoop Architecture, its uses and its components comprising of the MapReduce framework [1], Hadoop Distributed File System as well as their subcomponents.

Apache Hadoop is an open source software framework written in Java which is used to process vast amounts of data in parallel. The Hadoop cluster provides a parallel computing cluster that is partitioned across many servers to provide scalability. It is designed to bring computing to the data. Each DataNode holds part of the data and should be able to process the data which it holds. The programming model used in Hadoop is the Map Reduce model. Scalability, fault tolerance, and speculative execution are some of the features present in Hadoop

2.1. HADOOP ARCHITECTURE

Hadoop Cluster consists of a central node called the master node and several worker nodes also called the slave nodes. The master node accepts the job from the client and distributes it to the slave nodes. The master node runs three components a) NameNode b) Secondary NameNode c) JobTracker. The NameNode keeps the directory tree of all files in the file system, and keeps a track of the location in the cluster where the file data resides. When a client wishes to copy, add, delete or move a file, it interacts with the NameNode. The Secondary NameNode provides redundancy in case of failure of primary NameNode. It also contacts NameNode

[1] http://en.wikipedia.org/wiki/MapReduce

and takes checkpointed snapshot images which are used if the NameNode fails. The JobTracker distributes the map and reduce tasks to the nodes in the cluster and also controls the scheduling of the tasks on the nodes. Generally it distributes tasks in a data local manner to nodes which already have the data for that task.

The slave nodes run the components a)DataNode b)TaskTracker. The DataNode runs the Hadoop Distributed File System (HDFS) [2] and contains a portion of the data in HDFS. It also keeps a connection with the NameNode which provides it with the file system operation needed to be performed. The TaskTracker accepts the tasks be it map or reduce and keeps track of the status of each task until they finish. Each TaskTracker is configured with a set of slots which is an upper limit on the number of tasks it can accept in any given time. The TaskTracker notifies the job tracker on successful completion of a job and also if the task failed for any reason.

Uses The primary use of Hadoop is as a computing platform to analyze as well as process huge amount of data. It also provides HDFS to store large amount of data in a distributed way. It also provides components which enable the data inside HDFS to be presented in an interface similar to SQL. This makes the access to the data more streamlined and also to make integration with existing systems easier. Any application which scales well and which requires minimum communication can exploit the distributed processing capability of hadoop

2.2. MAPREDUCE

Input and Output types of a MapReduce job:

(Input) <k1, v1> ->map -><k2, v2> ->combine -><k2, v2> ->reduce -> <k3, v3>(output)

The map takes an input split, does processing on it based on the mapper algorithm

[2] $https://hadoop.apache.org/docs/r1.2.1/hdfs\_design.html$

and emits a key value pair. After the map comes the shuffle stage which copies and sorts the output on key and aggregate values. This key and aggregated values for it are sent to the reducer which outputs a reduced key value pair.

The map and reduce functions are provided by the application through implementations of appropriate interfaces and abstract classes. The job configuration is provided in the hdfs-site.xml, mapred-site.xml and core-site.xml. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

When the JobTracker is notified by the TaskTrackers heartbeat call of a task attempt that has failed, it will reschedule the execution of the task. The JobTracker does not reschedule the task on a TaskTracker on which it has previously failed. Also, if a task fails four times (or more), it will not be retried further.

2.3. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

Hadoop includes a distributed filesystem called Hadoop Distributed Filesystem (HDFS) . It is designed for storing large files of the order of petabytes with streaming data access running on commodity hardware. It follows the write once read many times and since analyses are performed over the whole dataset, time to read it should be very fast. The write once read many model relaxes concurrency control issues, makes data coherency simple and enables high throughput. Data writes are restricted to one writer at a time.

HDFS Architecture

In HDFS Architecture, the NameNode manages filesystem operations and maps data blocks to DataNodes. The DataNodes ask the NameNodes for the type of file operations to perform. The NameNode returns values to the functions called from the DataNode. The NameNode maintains and administers changes to the file system namespace.

Data replication and organization One of the key features of HDFS is that it provides fault tolerance. It does so by replicating file blocks. The number of replications can be provided by the user during input time. HDFS replication is rack aware so as to use network bandwidth intelligently. The location of the DataNodes is identified by the NameNodes through the rack IDs.

The size of a HDFS block is 64 MB. Each file consists of one or more of these blocks. Typically HDFS tries to place the blocks in separate DataNodes. When a client creates a file, HDFS caches the data to a temporary file and goes on filling it with data until it reaches 64 MB. Once it does, the file is converted to a block and stored in a DataNode and the NameNode is notified of the block. The temporary file is then destroyed by HDFS.

Failure detection and prevention Failures in HDFS can occur in NameNodes, DataNodes or network connectivity problems. HDFS uses heartbeat messages to detect the presence of connection between NameNode and DataNode. Each DataNode sends messages to the NameNode indicating it is alive. If the NameNode stops receiving the message from the DataNode, it marks it as dead and stops sending it requests. Since the dead node no longer responds to messages, hence the data present in that node is considered to be lost. If the loss of a node causes the replication factor to go below the minimum value, the NameNode starts the process of replicating the lost data in other nodes.

Another technique employed by HDFS to prevent loss of data due to failure is by rebalancing the data blocks based on different models. The main considerations while doing data rebalancing is that there should not be data loss due to rack failure, network I/O should not be increased and the data should be spread uniformly throughout the cluster

Data integrity HDFS employs checksum validation to ensure data integrity across a cluster. It computes checksums on files and stores them in hidden files in the same namespace as the data. Thus the client can compare the checksums to see if the data it receives is correct and not corrupted.

File permissions HDFS provides read, write and execute permissions on the data that it stores. The read and write have their normal meanings while the execute attribute corresponds to permission for accessing a child directory of the parent directory.

Chapter 3

CURRENT HADOOP SCHEDULERS

In this chapter we describe the current schedulers present in Hadoop, their key features, as well as the job profiles and the type of organization setting each scheduler best fits into.

Hadoop was designed to process large batch processing jobs like mining of log files and indexing of the web. The Hadoop FCFS scheduler was suitable for processing such jobs. However, to run the clusters efficiently, a MapReduce cluster should be shared with multiple users so as to increase system utilization. Incorporating such a feature into Hadoop requires cooperation from the scheduler to allocate resources among users fairly, provide good response time as well as guarantee certain amount of resources for each user. Thus Fair Scheduler, developed by Facebook and Capacity Scheduler by Yahoo are two additional pluggable schedulers available for the Hadoop framework.

3.1. FIRST COME FIRST SERVER (FCFS SCHEDULER)

In FIFO scheduler, the nodes are assigned to tasks in the order in which they make requests. The scheduler has a single queue with which it schedules the tasks with the oldest task chosen first.

3.2. FAIR SCHEDULER

In this scheduler [1], the resources (nodes) are assigned to jobs such that each of them get equal share of the resources over time. The fair scheduler organizes jobs

[1] *https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html*

into pools, and divides resources fairly between these pools and pools overlap over resources. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster. Each pool gets it guaranteed minimum share, so that those users or groups get a minimum share of the resources. Resources not used by a pool are split among the other pools. The scheduler also allows a pool to preempt jobs in other pools if it's guaranteed minimum share is not met and also limits the number of concurrent jobs which a user and a pool can run.The scheduler also includes a number of features for ease of administration, including the ability to reload the configuration file at runtime to change pool settings without restarting the cluster, limits on running jobs per user and per pool, and use of priorities to weigh the shares of different jobs.

3.3. CAPACITY SCHEDULER

Capacity Scheduler [2] maximizes the throughput and utilization of the cluster on which they run the MapReduce jobs. Capacity Schedulers enable sharing of the cluster between multiple users so as to have good average utilization and also provide cost effective elasticity among users to access excess capacity left idle by other users. It provides a limit on each user so that a single user does not consume more than its allocated share of resources. It provides multi-tenancy and also provides safe-guards to ensure that privacy and security is maintained among the different users sharing the same cluster. The queues also support job priority where higher priority jobs have access to the resources before jobs with lower priority. The capacity scheduler tries to simulate a separate FIFO/priority cluster for each user, rather than performing fair sharing between all jobs. It configures a wait time on each queue after which it is allowed to preempt tasks of other queues.

[2] *https://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html*

In a nutshell Fair Scheduler is used in small as well as large clusters. It works best for diverse jobs, providing good response time to both small and large jobs. The Capacity Scheduler works well when there are multiple clients running multiple types of jobs with different priorities as it allows for reuse of unused capacity and priority of jobs in queues.

For our purpose we enhance the FCFS scheduler to incorporate thermal awareness as our thermal aware scheme as discussed in the next chapter fits best into it.

Chapter 4

THERMAL AWARE SCHEDULER

In this chapter we begin by describing our system model. Then we briefly explain the general layout of the data center as well as the cooling energy associated with a typical data center. Next, we present an overview of thermal aware scheduling algorithms proposed in literature. Finally, we present our scheme which extends the existing thermal aware scheduling algorithms for Hadoop MapReduce.

4.1. SYSTEM MODEL

We assume a data center with non-uniform temperature distribution within the data center room. Depending on data center layout servers have non-uniformly contribution on the heat recirculation in the data center room . Further, we assume the data center runs data-processing jobs using map-reduce frameworks. In particular, we focus on the Hadoop implementation of the map-reduce framework. Finally we assume the data-processing jobs consist of delay-tolerant batch processing jobs, such as background log analysis, that do not have strict completion time requirements; they can be delayed by a bounded amount of time. Cooling energy savings are possible by being able to temporally "spread" the workload, and assign it to the computing equipments which reduce the heat recirculation in data center room and therefore the load on the cooling systems. In the following section we give an overview on the data center cooling energy model.

4.2. DATA CENTER LAYOUT AND COOLING ENERGY

We consider typical air-cooled data centers, in which the servers are arranged in chassis which are in turn arranged in racks with the racks being organized in rows. In each aisle, between two rows, the front panels or the rear panels face each other and are termed cold/hot aisle respectively. The computing nodes or chassis consume power and generate heat according to the amount of power they consume. Computing Room Air-Conditioners (CRAC) provide the cooling through the perforated tiles in the floor Moore *et al.* (2005). Figure 4.1 shows a typical data center layout. The cold and hot aisles are alternating with the inlet side of the server faces a cold aisle and the outlet side faces a hot aisle. Cool air is blown from the CRAC which passes through the perforated tiles placed in the cold aisles to the inlet of the servers. The heated air is then returned to the CRAC. Ideally all the HOT air should directly go back to the CRAC; but, in practice, some of the hot air recirculates back to the computing nodes. Figure 4.1 shows that improper thermal management can create HOT spots within the floor as a result of the intermixing of the hot and cold air. These hotspots increase the inlet air temperatures of the individual server racks. To control the HOT spots the air conditioning is set to low temperatures which in turn impacts the energy efficiency of the data centers.

Usually, the heat recirculation is not uniform, i.e., different parts of data center have different contribution in the heat recirculation.

The temperature of the supplied cooled air by CRAC, denoted as T_{sup}, should be low enough so that the inlet temperature of none of the computing nodes does not go beyond the red line temperature (T_{red}) which is specified by the manufactures. Due to heat recirculation in data centers, the inlet temperature vector of servers, denoted

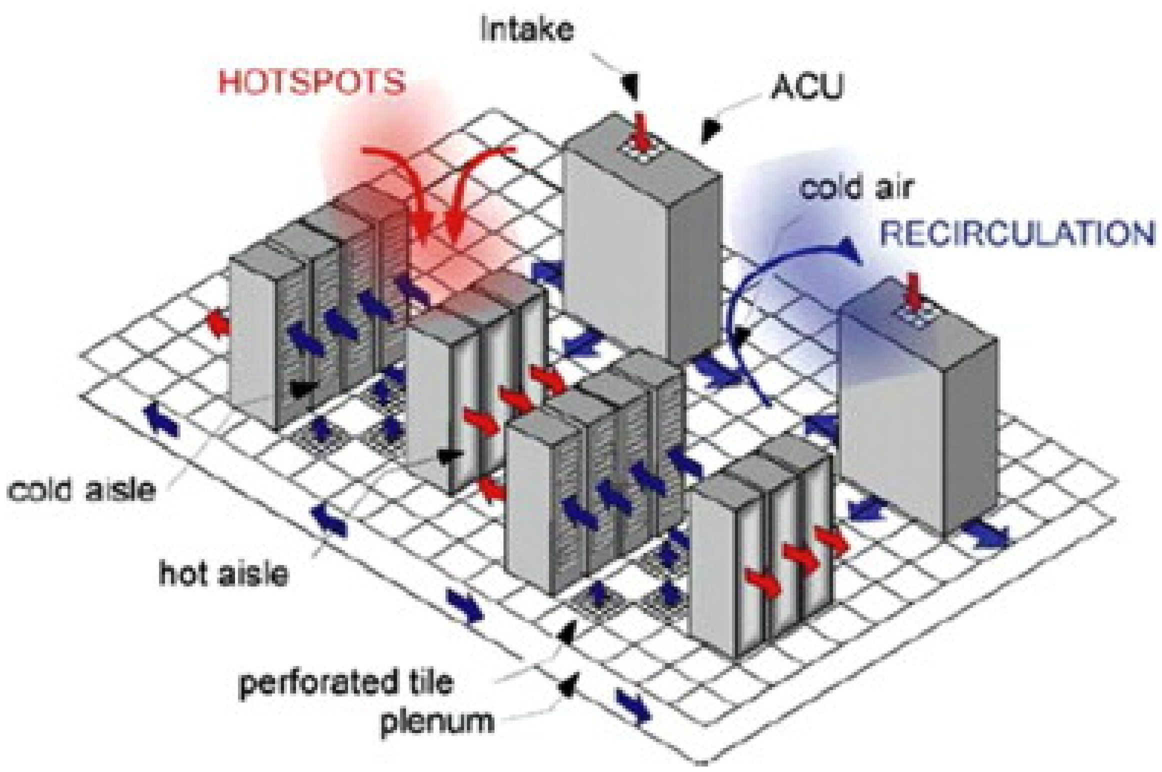


Figure 4.1: Datacenter Layout

by T_{in} can be written as:

$$T_{in} = T_{sup} + T_{rise}, \tag{4.1}$$

where T_{rise} denote the temperature rise of servers due to heat recirculation in data center room (the heat that the servers receive from itself and all other systems in data center room. Eq. 4.1 shows that the inlet temperatures of servers depend on the power consumption of nodes (since T_{rise} is directly affected by the nodes' power consumption), and consequently on the amount of workload they are assigned. On the other hand, cooling energy depends on T_{sup}. Cooling energy of the CRAC can be modeled by its *coefficient of performance* (CoP), which is the ratio of the heat removed (i.e., computing energy) over the work required to remove that heat (i.e., cooling energy). A higher CoP means more efficient cooling, and usually the higher the required supplied temperatures (T_{sup}), the better the CoP is. In other words, CoP is usually monotonically increasing function of the supplied temperature, e.g., for an HP data center CoP reported as $CoP(T_{sup}) = 0.0068T_{sup}^2 + 0.0008T_{sup} + 0.458$ Moore *et al.* (2005). However, according to Eq. 4.1, the highest CRAC output temperature is limited by the servers' *redline temperature*. Therefore, T_{sup} can be *at most* equal to:

$$T_{sup} = T^{red} - \max(T_{rise}), \tag{4.2}$$

where the function max ensures that the supplied temperature of CRAC does not exceed the redline temperature of the hottest equipment. Respectively, the cooling power, denoted by P^{AC}, can be written as a function of the CoP of the supplied temperature:

$$P^{AC} = \frac{P^{comp}}{CoP(T^{red} - \max(T_{rise}))}, \tag{4.3}$$

where P^{comp} denotes the total computing power. Eq. 4.3 suggests that for *a given load*, the cooling power can be potentially improved by efficient workload distribution. Intuitively, this is possible if the workload distribution is thermally balanced among the servers, meaning that a higher portion of the workload is assigned to the servers that have the least contribution on the heat recirculation and consequently on the maximum T_{rise}. We refer to those servers as *thermally efficient* servers. In the following we give an overview on the existing thermal aware scheduling algorithms which try to decrease cooling energy by removing HOT spots, i..e, minimizing the maximum T_{rise}.

4.3. OVERVIEW OF THERMAL AWARE SCHEDULING IN LITERATURE

There are considerable amount work which try to develop efficient thermal aware scheduling algorithms for data centers. The algorithms are different depending on (i) the workload type, (ii) the optimality and the complexity of the solution, and (iii) the data center thermal input parameters. Particularly, we divide the algorithms depending on their thermal input parameters as (i) algorithms which need the complete heat recirculation model such as TASP and XInt-GA, and (ii) the algorithms which use temperature/or a summary of data center thermal model such as inverse-temperature and MinHR.

- **XInt-GA and XInt-SQP:** XInt-GA and XInt-SQP are based on the heat recirculation coefficient for all pairs of nodes in a data center, considering the data center layout and thermodynamic conditions:

  $$D = \{d_{i,j}\}_{N \times N}$$

  Where, N is the total number of nodes and each $d_{i,j}$ denotes the fraction of heat that flows from node j to node i Tang *et al.* (2007). D is referred to as the heat circulation matrix and can be calculated according to data center air flow characteristics and data center layout. Tang *et al.* (2008) propose XInt-GA and XInt-SQP to minimize peak inlet temperatures leads to lowest cooling power needs. They formulate the problem as a minimization of peak inlet temperature through task assignment and solve it using XInt-GA, a genetic algorithm (meta-heuristic) approach as well as XInt-SQP, a sequential quadratic approach.

- **TASP-MIP, and TASP-LRH :** Abbasi *et al.* (2012) propose Thermal Aware Server Provisioning(TASP) which decreases the heat generated in the servers through server provisioning in Internet data centers. Similar to Xint-GA and Xint-SQP, TASP uses heat recirculation model of the data center as input. The authors, formulate the problem as choosing the active server set among a set of servers so as to minimize total energy. The authors solve this problem using Mixed Integer Programming (i.e., TASP-MIP) and propose a and N-approximation greedy algorithm (i.e., TASP-LRH).

- **MinHR:** MinHR proposed by Moore *et al.* (2005) is a power provisioning policy that minimizes amount of heat recirculating inside data center. Their goal was to minimize total amount of heat recirculating the data center before returning to the CRAC and to maximize power budget and utilization of each server. The algorithms uses the Heat Recirculation Factor (HRF), as input parameter. HRF models the total heat recirculation of one chassis on all other servers (i.e., for a data centers with whose servers has homogeneous power consumption, HRF for each server is equal to the sum of the correspond row of heat recirculation matrix D). MinHR is a heuristic solution which ranks chassis based on the ratio of the HRF of each individual chassis to the sum of all HRFs, and assign tasks to the chassis with lowest HRF ratio.

- **Inverse-temperature:** This algorithm proposed by Sharma *et al.* (2005) where imbalances in temperature are resolved heuristically by redistributing workload inversely proportional to the chassis outlet temperature. Thermal load balancing is achieved by providing cooling inlet air for each rack below redline temperature, maintain uniformity in inlet temperature and also dynamically responding to thermal emergencies that cause uneven temperature.

We can see that all of the above algorithms introduce a solution to choose thermally efficient servers. We aim to use Inverse-temperature algorithm in classifying servers as thermal efficient for use in Hadoop

4.4. PROPOSED THERMAL AWARE SCHEDULER (TAS)

Hadoop workload is typically batch processing and not very delay sensitive in the sense that a fair proportion of the jobs are delay tolerant. It is comparable to HPC workloads which are also delay tolerant. In HPC workloads jobs are distributed to the servers whereas in Hadoop, servers request the scheduler for a task to execute,

In the current implementation, Hadoop scheduler performs as follows:

1. Heartbeat Message: The heartbeat message is the means of communication between the TaskTracker and the JobTracker. The key fields which this message contains maximum Map and Reduce tasks,total physical and virtual memory, frequency of CPU and CPU time, responseid, state of TaskTracker health. The heartbeat message is sent periodically be the TaskTracker to the JobTracker and if the JobTracker does not get a heartbeat message from the TaskTracker for an interval of time, then it labels that node as unhealthy and allocates the tasks given to that TaskTracker to other healthy TaskTrackers.

2. JobTracker: The JobTracker receives the heartbeats and then processes them to extract the information for the TaskTracker. It process the delay, checks if it has reached zero. If it does then it schedules a task on that TaskTracker otherwise it decrements the delay value and sends it with the reply heartbeat message to the TaskTracker.

3. TaskTracker: The TaskTracker executes the task on its node. There are roughly four categories of tasks 1. Setup task 2. Map Task 3. Reduce Task 4. Cleanup task. Setup task is to check if resources are available and if so then it assigns a task to a slot. Map task takes a list of data elements are provided, one at a time, to a function called the Mapper, which transforms each element individually to

an output data element. Reducing task lets you aggregate values together. A reducer function receives an iterator of input values from an input list. It then combines these values together, returning a single output value. Cleanup task frees the allocated resources for a particular task to make it available for subsequent tasks.

We target to add thermal awareness to this scheduler, by postponing the jobs as much as possible so that they are assigned to the thermal efficient servers. The problem statement in our project is as follows

Given a map-reduce job to distribute among slave nodes (as map or reduce tasks), and a maximum delay, denoted by d_{max} that each task of map-reduce can be postponed, what is the task assignment on the servers, so that the hotspot temperature is minimized?
Depending on the nature of tasks and the data center thermal model, this problem can be framed as an optimization problem which leads. However to to a complexity of such a solution (thermal aware scheduling is proven to be NP-hard problem, we propose a heuristic solution as follows:

We propose to classify servers according to their thermal efficiency as "HOT", "MEDIUM" or "COLD". This classification can be done efficiently using the existing heuristics thermal aware scheduling algorithms(i.e., MinHR, inverse-temperature, and TASP-LRH). We disregard Xint-GA and Xint-SQP and TASP-MIP due to their high time-complexity.

4.4.1. Challenges in incorporating thermal awareness in Hadoop Scheduler

The Hadoop scheduler gives a task to a node for execution only when the node requests for a task from it. The scheduler itself does not decide which node to give a particular task to. In such a model, the main challenge was to figure out how to allocate more tasks to thermal efficient servers i.e. servers that contribute least to cooling energy and one way was to make thermally inefficient servers wait for an interval of time, thus indirectly giving them less tasks to execute. The second challenge was in determining the delay interval for the thermally inefficient servers. The delay interval can be decided based on the deadlines for a job.

In data distribution performed by HDFS, the whole file is broken into blocks and the blocks are distributed among the servers incorporating redundancy in them. To incorporate redundancy and fault tolerance HDFS distributes the blocks in a rack aware fashion if it knows the racks. However within a rack the data is distributed randomly. As future work, the distributed of data within a rack can be done based on the thermal profile of the servers.

The nodes classified as “COLD” are given tasks by the JobTracker instantaneously as they have less effect on the heat recirculation. Those nodes classified as “COLD” are allocated a wait-time hereby referred to as “delay”. This “delay” is a factor of the total running time of the job. When the JobTracker distributes the tasks to the TaskTracker, nodes classified as “HOT” start decrementing the delay value till it reaches zero. Once the delay value has reached zero, then if the job has not yet completed and tasks are still left, then these “HOT” nodes enter the pool of nodes which can be allocated a task. Thus in effect, “COLD” nodes are preferentially allocated tasks over the HOT servers so as to reduce the cooling energy costs.

The TaskTracker running on the slave nodes periodically sends heartbeat messages to the JobTracker to inform it that it is still alive and also whether it is ready to run a new task. To this heartbeat, the JobTracker sends its own heartbeat consisting of responseID, interval for next heartbeat as well as all the information required for the TaskTracker to run a new job. The heartbeat message was modified to incorporate a new field named as delay. The delay value of the "COLD" nodes was set to zero while the delay for the "HOT" nodes was set to a user-defined value. This ensured that the COLD nodes would be allocated tasks preferentially over the "HOT" nodes. The value of delay for the HOT nodes is determined by the workload characteristics. If the workload is delay-intolerant or delay sensitive then the delay value is set to a lower value so as to provide maximum amount of available resources to the job so as to enable it to finish quickly. However if the reverse is true then delay is set to a higher value to maximize the amount of task processing performed by the "COLD"nodes. Thus we see that delay acts as a tunable parameter indicating the trade-off between cooling costs and delay.

Every time the TaskTracker sends a heartbeat message, the JobTracker receives it, decrements the delay parameter and sends it back to the TaskTracker. When the delay becomes zero, then the node is free to accept a task if there are tasks still left to process. Figure 4.2 presents a broad overview of the system with respect to the Hadoop framework.

Figure 4.3 shows a state machine diagram describing the states which a task tracker transitions based on the delay parameter. Once the job has been accepted, if the delay is more than zero, then, the TaskTracker gets its delay value decremented every time it asks for a task from the JobTracker through the heartbeat message. While doing this it stays in the idle state. When its delay becomes zero it accepts a task, transitions to the running state and upon finishing it again asks for a task. If there are tasks available, then it repeats the above process otherwise it transitions to the stop state.

If the number of tasks is less than the number of servers then "cold" servers have priority in executing those tasks and hence by the time the "delay" period is over, the job would have finished executing thereby introducing no "delay" in the total

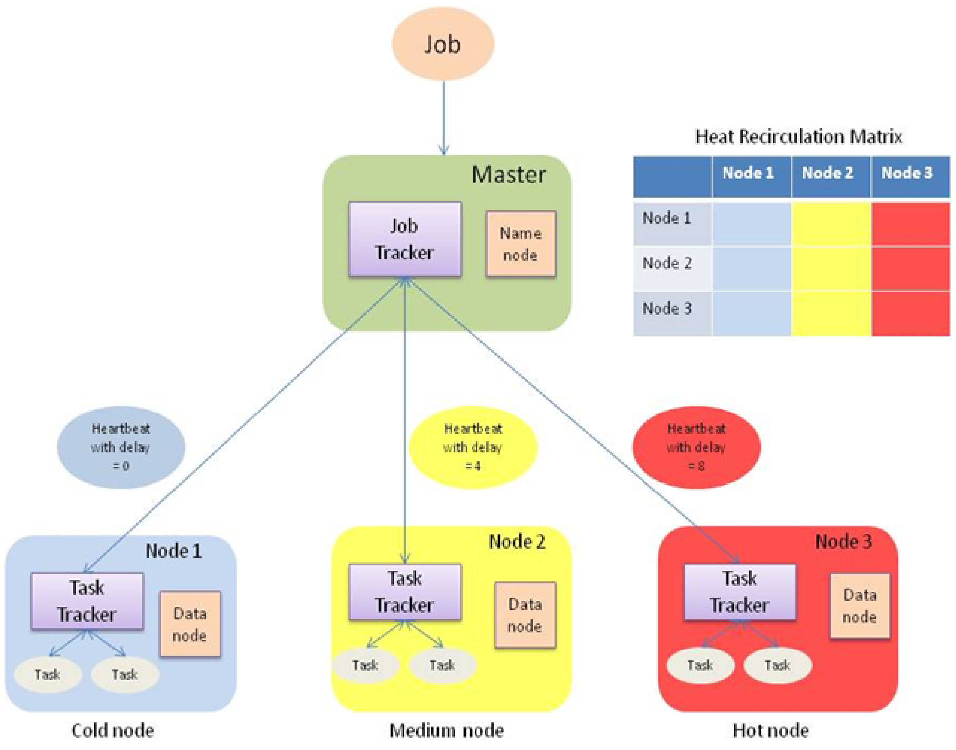


Figure 4.2: Overview of proposed scheme incorporating heat recirculation matrix

runtime of the job.

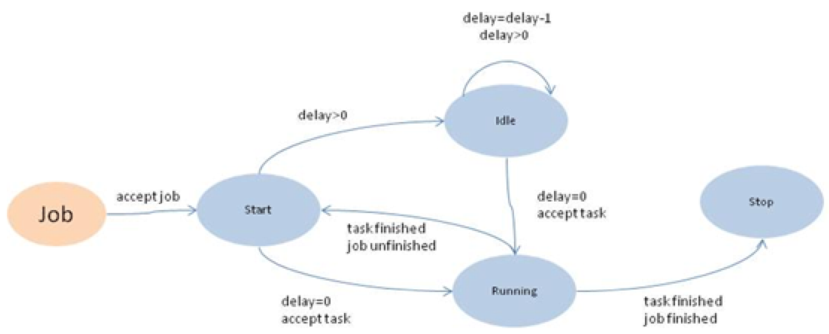


Figure 4.3: State Transition diagram of a TaskTracker.

Algorithm 1 Steps for Thermal Aware Scheduler

```
1: Classify servers as "HOT", "MEDIUM" or "COLD" based on initial outlet temperature using existing
   scheduling algorithms (e.g., MinHR, TASP-LRH, inverse-temperature).
2: Take as input the job from the client
3: Break up the job into Map and Reduce tasks
4: Assign delay value to the TaskTracker of the nodes based on its classification (zero delay for "COLD"
   nodes, $d_med$ to "MEDIUM" nodes and $d_{max}$ to "HOT" nodes.
5: for each request from the TaskTracker do
6:     For every subsequent request by the TaskTracker of that node for a task, check if it's delay value
   equals zero
7:     If non zero then decrement it's delay count by one and listen for subsequent request from the
   TaskTracker of that node
8:     If the delay is zero, schedule that task on that TaskTracker if tasks are still left to be scheduled
9: end for
```

Chapter 5

RESULTS AND ANALYSIS

This chapter evaluates our Thermal Aware Scheduler (TAS) compared with the current FCFS scheduler of Hadoop. We first, describe the test setup comprising of the hardware and software configuration used to build the Hadoop cluster. Next we describe the type of benchmarks, their characteristics and features, which we use to compare TAS with the current FCFS scheduler. We also explain in tabular fashion the parameters which we used to vary to make our test more diverse. To evaluate TAS compared to FCFS we measure the HOT spot temperature (i.e., peak temperature of servers) and the servers' average temperature for different runs of TAS and FCFS with CPU-intensive and I/O intensive workloads and different delay intervals. Further, we also present the detail of the temperature profile of all of the servers when running FCFS and TAS.

5.1. EXPERIMENTAL SETUP

The Hadoop cluster was setup on nine blade servers with secure shell access. The temperature was measured using the "ipmitools" available in Linux. All the servers are hosted in BlueTool data center and the power was measured by monitoring the PDU's of each rack in the data center

5.2. HARDWARE SETUP

The blade servers have homogeneous composition in their hardware. All nodes have 10/100 Mbps Ethernet interface and the whole rack is connected to a 100 Mbps Dell Network Switch. The total HDFS size was 500 GB. Table 5.1 illustrates the hardware setup in detail

Table 5.1: Hardware Configuration of Servers

Node	RAM	Disc Storage	Processor
Intel	8 GB	80 GB	Intel Xeon 3.6 GHz.

5.3. SOFTWARE SETUP

Hadoop version 1.0.3 was installed in all of the nodes along with Java 1.6 as well as "ipmitools" to measure temperature. Each of the servers are connected to each other through passwordless ssh. Ubuntu 12.04 was the operating system for all the nodes. The monitoring framework consists of a python script that monitors the temperature of the servers with the "ipmitools" suite. Power is read from the Power Distribution Unit using "SNMP" protocol. The interval for the readings and the servers to monitor is given as input to the monitoring script.

5.4. COOLING ENERGY SAVINGS WITH WORDCOUNT WORKLOAD

Hadoop was configured with the default settings and with a data replication factor of 3. A sample workload of Wordcount, which counts occurrence of each word, was run to identify the CPU utilization of the DataNode and TaskTracker in the slaves. During job runtime, the average CPU utilization of the DataNode was 2 % while that for the TaskTracker was found to be 75 %. This shows that CPU utilization and hence the temperature rise is due to the task execution by the TaskTracker and not due to the other components of Hadoop. The MapReduce job was broken up into six tasks. The cooling power consumption as shown in Figure 5.1 shows that when the incoming tasks are considerably less than the number of servers, i.e. capacity is more than demand, TAS allocates the tasks to the "COLD" servers first thereby decreasing the cooling power and cooling energy, whereas FCFS allocates the tasks randomly to each servers. Table 5.2 shows the total cooling energy consumption with the FCFS and TAS schedulers and shows that FCFS scheduler consumes 846.1 joules whereas TAS consumes 844.8 joules

Table 5.2: Cooling power for Wordcount workload with FCFS and TAS scheduler

Time(s)	Cooling power(W) with FCFS	Cooling power(W) with TAS
0	65.1159	65.2687
1	65.1159	65.2688
2	64.8103	64.8103
3	64.8103	64.9631
4	65.2687	65.1159
5	65.4215	64.9631
6	65.1158	65.1158
7	65.1160	65.1159
8	64.9631	64.6575
9	65.2687	65.1159
10	65.2687	64.8103
11	65.1160	64.9632
12	64.6574	64.8103
Cooling Energy	FCFS 846.1	TAS 844.8

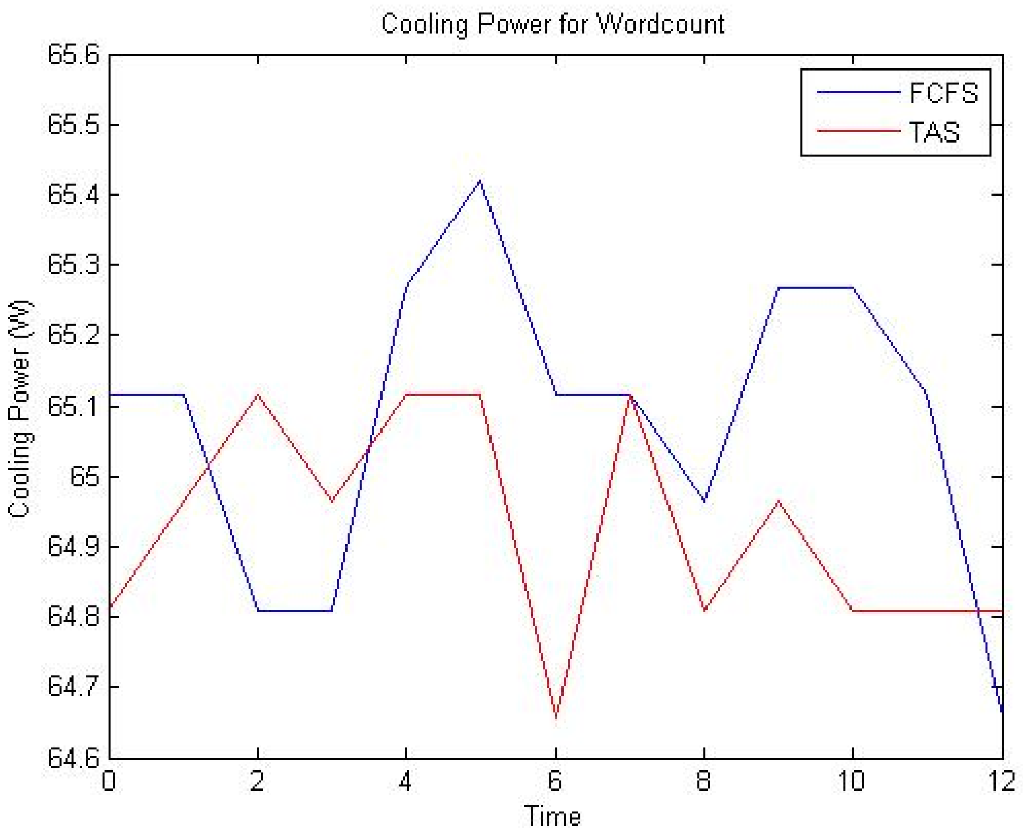


Figure 5.1: Cooling power comparison for Wordcount

5.5. BENCHMARKS

For our purpose we used the following benchmarks for running the schedulers

- PageRank workload of Intel Hibench Suite which is CPU intensive. PageRank is part of a benchmark suite developed by Intel called HiBench. PageRank workload represents large-scale web search indexing systems. Such a workload is typical for Hadoop MapReduce framework. It is an implementation of the page-rank algorithm which is used in web search engines as a link analysis algorithm. The PageRank workload was run on both Hadoop with FCFS scheduler as well as TAS. Figure 5.2 describes the PageRank algorithm.[1] The websites are

[1]http://www.math.cornell.edu/ mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

ranked based on how many websites link to it as well as how many websites a particular website links to. Based on this information, a directed adjacency graph is created which is then weighted based on how many sites link in and out of it. From the adjacency graph, the Adjacency matrix is then created and is then transposed to form the matrix A. Solving the matrix equation reveals the final rank of the websites.

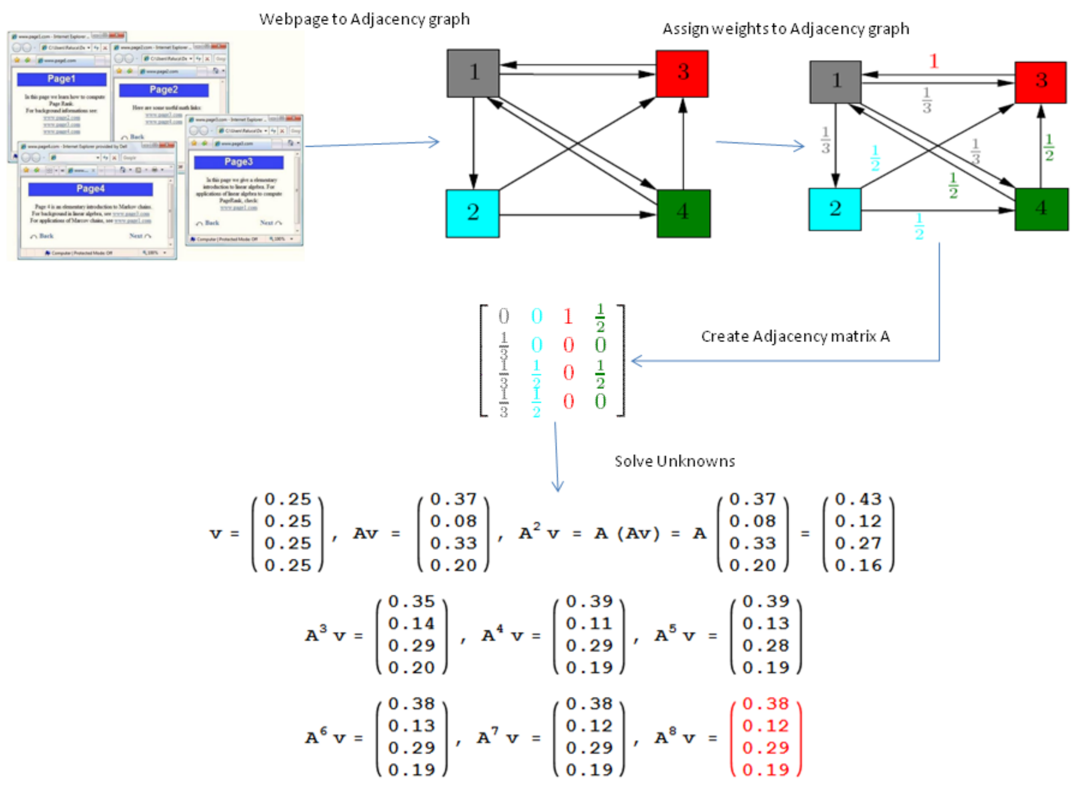

Webpage to Adjacency graph
Page1
Page2
Page3
Page4
Assign weights to Adjacency graph
1
2
3
4
Create Adjacency matrix A
$\begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$
Solve Unknowns
$\mathbf{v} = \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}, \mathbf{Av} = \begin{pmatrix} 0.37 \\ 0.08 \\ 0.33 \\ 0.20 \end{pmatrix}, \mathbf{A}^2\mathbf{v} = \mathbf{A}(\mathbf{Av}) = \mathbf{A}\begin{pmatrix} 0.37 \\ 0.08 \\ 0.33 \\ 0.20 \end{pmatrix} = \begin{pmatrix} 0.43 \\ 0.12 \\ 0.27 \\ 0.16 \end{pmatrix}$
$\mathbf{A}^3\mathbf{v} = \begin{pmatrix} 0.35 \\ 0.14 \\ 0.29 \\ 0.20 \end{pmatrix}, \mathbf{A}^4\mathbf{v} = \begin{pmatrix} 0.39 \\ 0.11 \\ 0.29 \\ 0.19 \end{pmatrix}, \mathbf{A}^5\mathbf{v} = \begin{pmatrix} 0.39 \\ 0.13 \\ 0.28 \\ 0.19 \end{pmatrix}$
$\mathbf{A}^6\mathbf{v} = \begin{pmatrix} 0.38 \\ 0.13 \\ 0.29 \\ 0.19 \end{pmatrix}, \mathbf{A}^7\mathbf{v} = \begin{pmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{pmatrix}, \mathbf{A}^8\mathbf{v} = \begin{pmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{pmatrix}$

Figure 5.2: PageRank Algorithm steps

- Teragen and TeraSort which is I/O (disc) intensive. TeraSort is one of the workloads available through the Hadoop examples jar package. Figure 5.3 shows the steps in the entire TeraSort workload. Teragen generates the random data which is saved in the directory called terain. TeraSort then sorts the random data using the key value paradigm of MapReduce. Finally the Teravalidate validates that the output from TeraSort step is sorted.

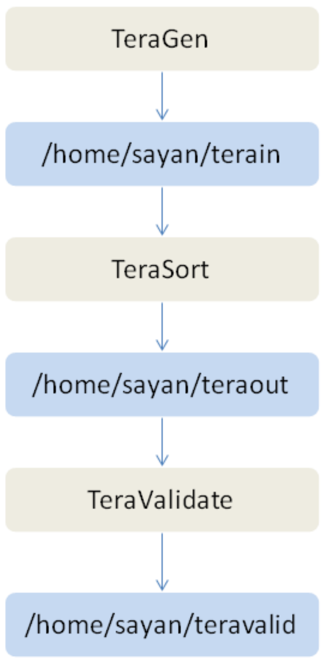


Figure 5.3: Steps of TeraSort workload

Table 5.3 below summarizes the benchmark characteristics as well as the parameters used to run the tests

Table 5.3: Workload Characterization Huang *et al.* (2010) and Test Parameters

Workload	Type	Job	Job/Map Input	Map Output	Reduce Output	Parameters
TeraSort	I/O	TeraSort	400 GB	45 GB	450 GB	Varying delay interval
PageRank	CPU	Dangling Pages Update-ranks SortRanks	1.21 GB 1.21GB 1.21GB	81.7K 5.3GB 86MB	30B 1.21 GB 167MB	Varying delay interval

5.6. TAS SETUP

The classification of the servers is performed based on the "inverse-temperature" scheme, meaning that one third of servers with highest temperature is classified as "HOT", and one third of servers with lowest temperature are classified as "COLD", and the rest are classified as "MEDIUM".

Also TAS is sensitive to the maximum delay (i.e., d_{max}) that any task can be postponed. For that we run TAS for two delay intervals, namely 'Large", and "Small" delay intervals. The magnitude of both delay intervals are chosen based on the total running time of the jobs.

5.6.1. TAS versus FCFS for CPU Intensive workload (PageRank) on Hadoop

We run experiments when using PageRank to evaluate TAS for CPU-Intensive workload. The PageRank workload takes about two and half hour to run. We run PageRank three times using FCFS, and TAS with two delay interval of large and small as described below. The maximum delay interval for postponing the tasks (i.e., d_{max} in Ch. 4, Algorithm 1) is chosen according to the running time of the jobs.

Table 5.4 shows the initial temperatures of the servers based on which they have been classified as either “HOT”, “COLD” or “MEDIUM” as well as the “Large” delay interval based on the number of requests. For Large delay experiment we assume tasks can be delayed for a long time; that is the delay interval for “HOT” servers is set 0.4 times the total running time of the job and was then converted into number of requests. For the “MEDIUM” servers the delay is set 0.2 times the running time of the job.

Table 5.4: Classification of Servers based on initial temperature with “Large” delay for PageRank workload

Server	Type	Initial Temperature (C)	Classification	Delay (requests)
9	Slave	37.5	MEDIUM	400
10	Slave	38.5	MEDIUM	400
11	Slave	43	HOT	750
12	Slave	43	HOT	750
22	Master	40	HOT	750
31	Slave	29	COLD	0
32	Slave	27	COLD	0
33	Slave	29	COLD	0
34	Slave	32	MEDIUM	400
35	Slave	39	HOT	750
36	Slave	35	MEDIUM	400

From equations 4.1, 4.2 and 4.3 we obtain the cooling power for every reading for both the TAS and FCFS schedulers. We use the power values for each server which multiplied with the Heat Recirculation Matrix for chassis we get the T_{rise}. Maximum of T_{rise} for all the chassis is then fitted into equation 4.3 to get the cooling power. Figure 5.4 shows the cooling power comparison between FCFS and TAS scheduler.

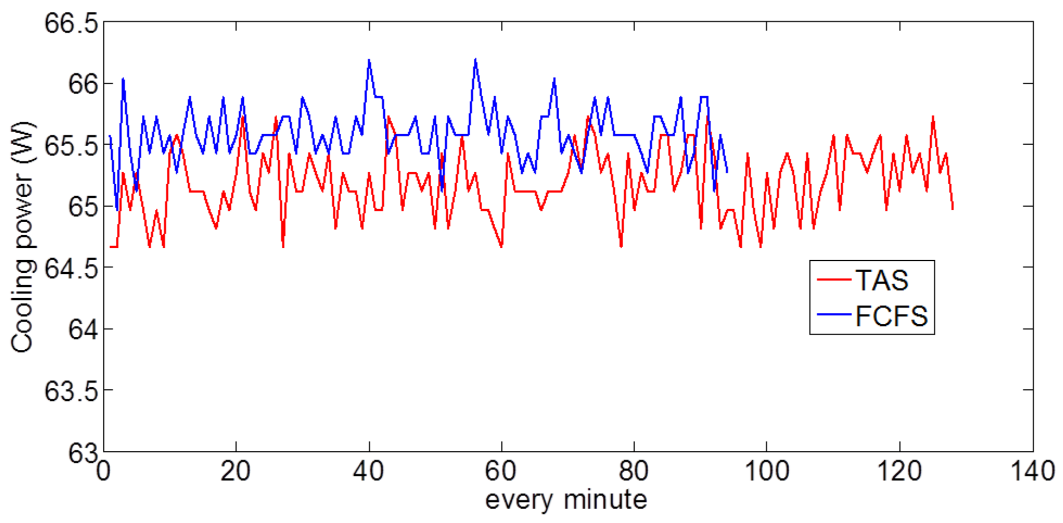


Figure 5.4: Cooling power for FCFS and TAS scheduler for PageRank workload with large delay

Table 5.5 shows the initial temperatures of the servers based on which they have been classified as either “HOT”, “COLD” or “MEDIUM” and the “Small” delay interval for each of the classification types. For the “HOT” servers the delay interval was 0.25 times the run time of the job while for the “MEDIUM” servers it was 0.15 times running time of the job.

Table 5.5: Classification of Servers based on initial temperature with “Small” delay for PageRank workload

Server	Type	Initial Temperature (C)	Classification	Delay (requests)
9	Slave	37.5	MEDIUM	220
10	Slave	38.5	MEDIUM	220
11	Slave	43	HOT	380
12	Slave	43	HOT	380
22	Master	40	HOT	380
31	Slave	29	COLD	0
32	Slave	27	COLD	0
33	Slave	29	COLD	0
34	Slave	32	MEDIUM	220
35	Slave	39	HOT	380
36	Slave	35	MEDIUM	220

We run PageRank three times using TAS according to the configuration given in Table 5.4 and Table 5.5, as well as when using FCFS.

Figure 5.5 shows the peak and average temperature of the servers with FCFS and TAS scheduler. The peak and average temperatures of the servers with TAS is considerably lower than with the FCFS scheduler. Also more the delay interval, greater is the reduction in peak and average temperatures achieved. The difference in the reduction in peak and average temperatures with TAS scheduler is more pronounced for the “HOT” and “MEDIUM” servers than the “COLD” servers. For certain “COLD” servers the average temperature seems to have increased due to the fact that they execute more tasks in TAS scheduler than they do in the FCFS scheduler.

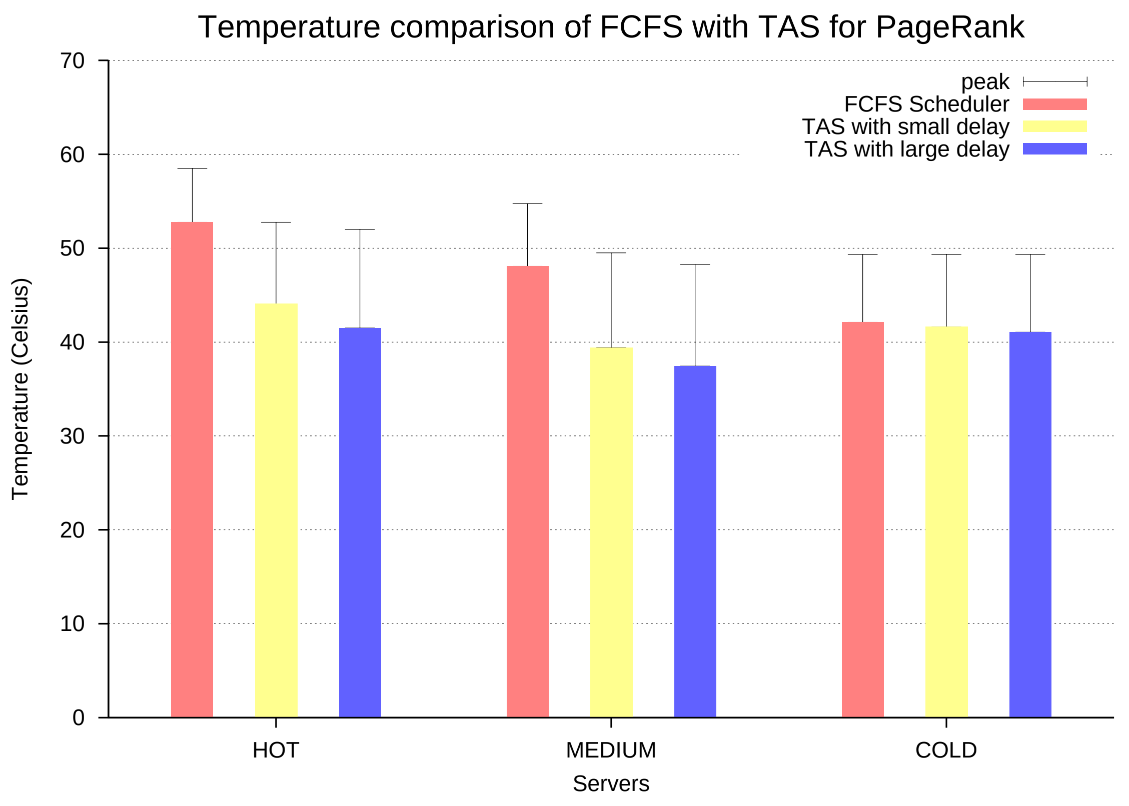


Figure 5.5: Peak and Average Temperature of FCFS compared with TAS with large and small delay

5.6.2. Temperature profile of "HOT" servers with "Large" delay

Figures 5.6,5.7,5.8 shows the corresponding temperature graphs when a server classified as "HOT" is run with the FCFS scheduler and the TAS. For Server 11,12,35 the graphs show that they run much cooler when run with TAS than with the native FCFS scheduler. Server 22 acts as the master and it also runs comparatively cooler with TAS than with the FCFS one. From Figure 5.9 we see that around 4:55, the temperature is around 44 but then shoots up to 52 once it starts executing the tasks.

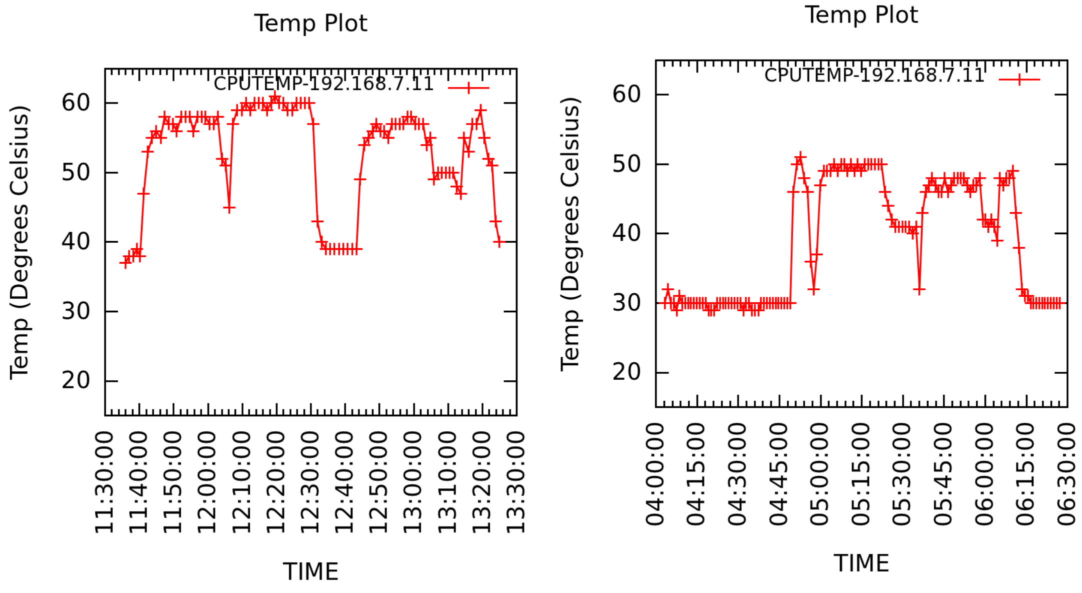


Figure 5.6: Temperature of Server 11 classified as "HOT" with FCFS(left) and TAS (right)

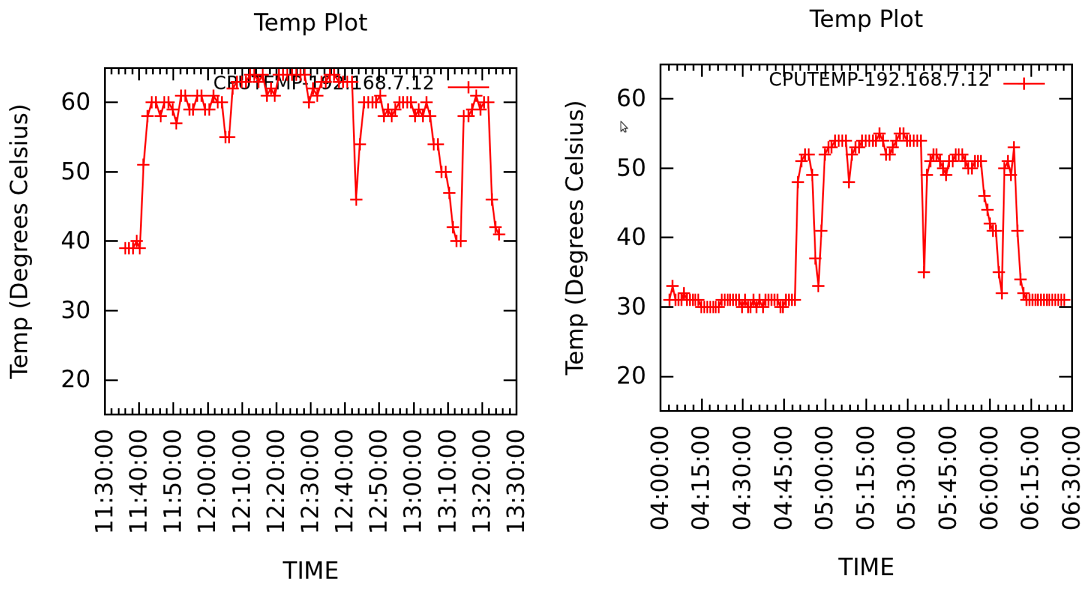


Figure 5.7: Temperature of Server 12 classified as "HOT" with FCFS(left) and TAS (right)

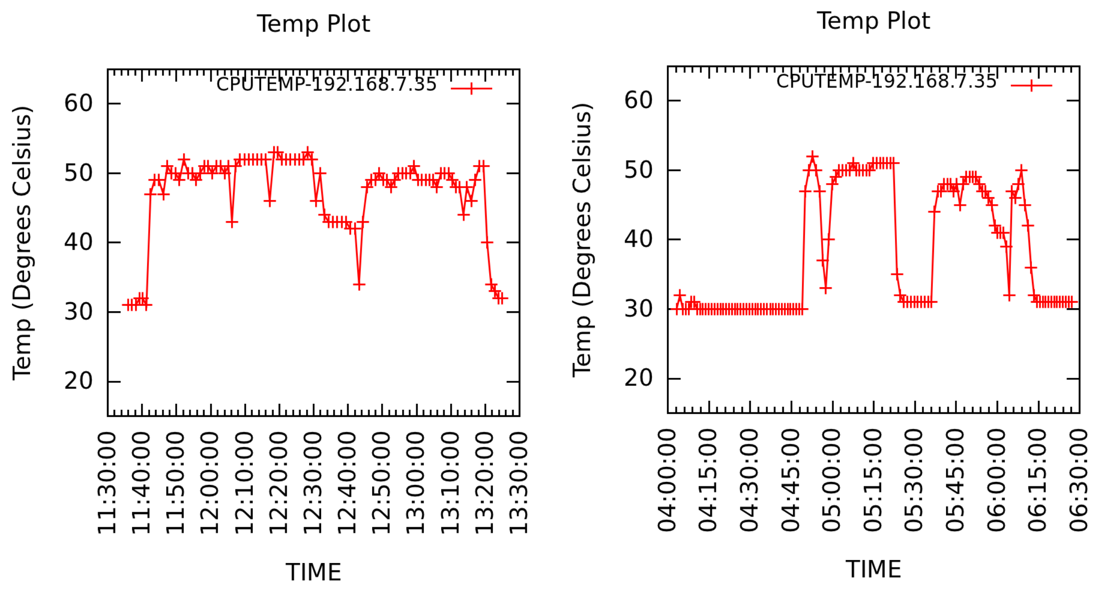


Figure 5.8: Temperature of Server 35 classified as "HOT" with FCFS(left) and TAS (right)

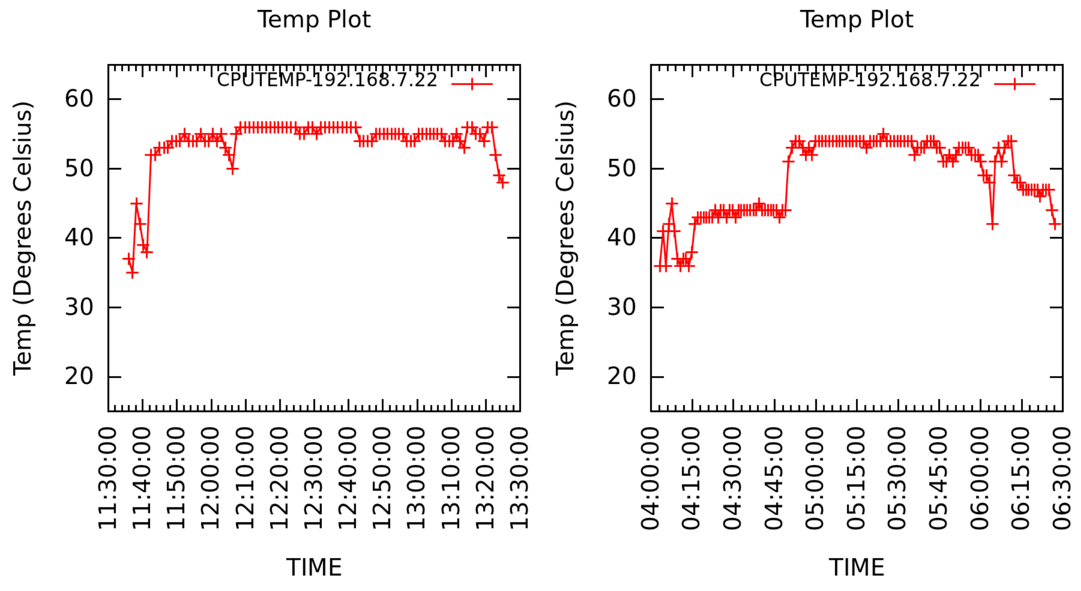


Figure 5.9: Temperature of Server 22 classified as "HOT" with FCFS(left) and TAS (right)

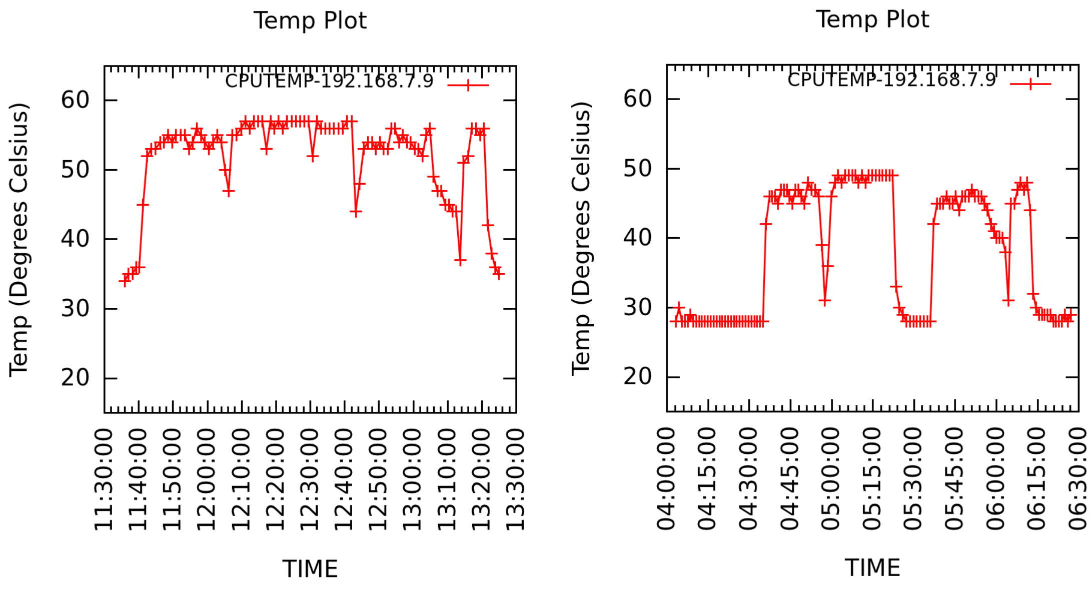


Figure 5.10: Temperature of Server 9 classified as "MEDIUM" with FCFS(left) and TAS (right)

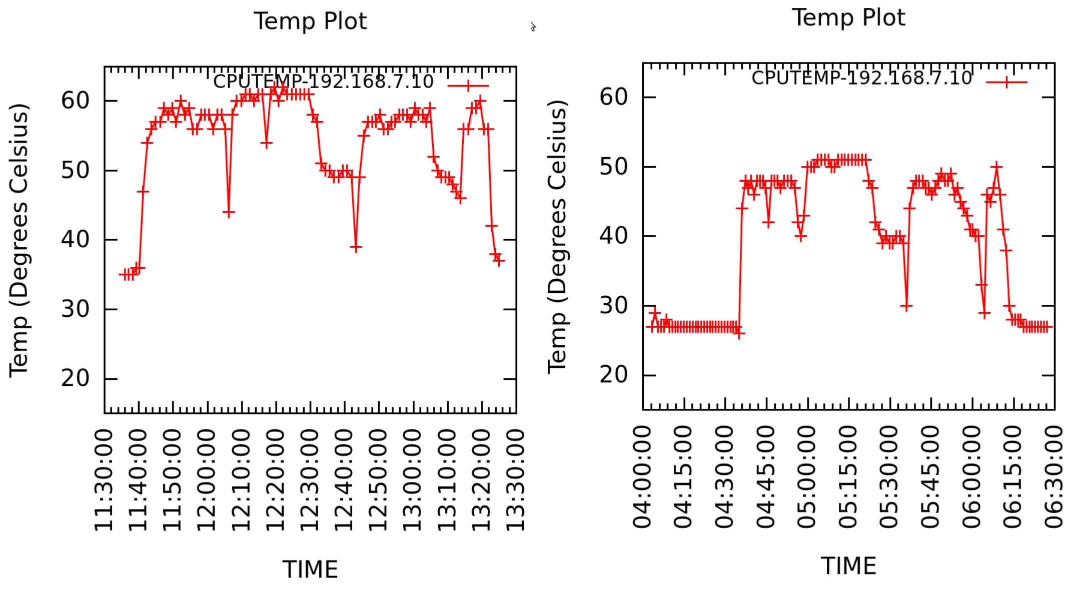


Figure 5.11: Temperature of Server 10 classified as "MEDIUM" with FCFS(left) and TAS (right)

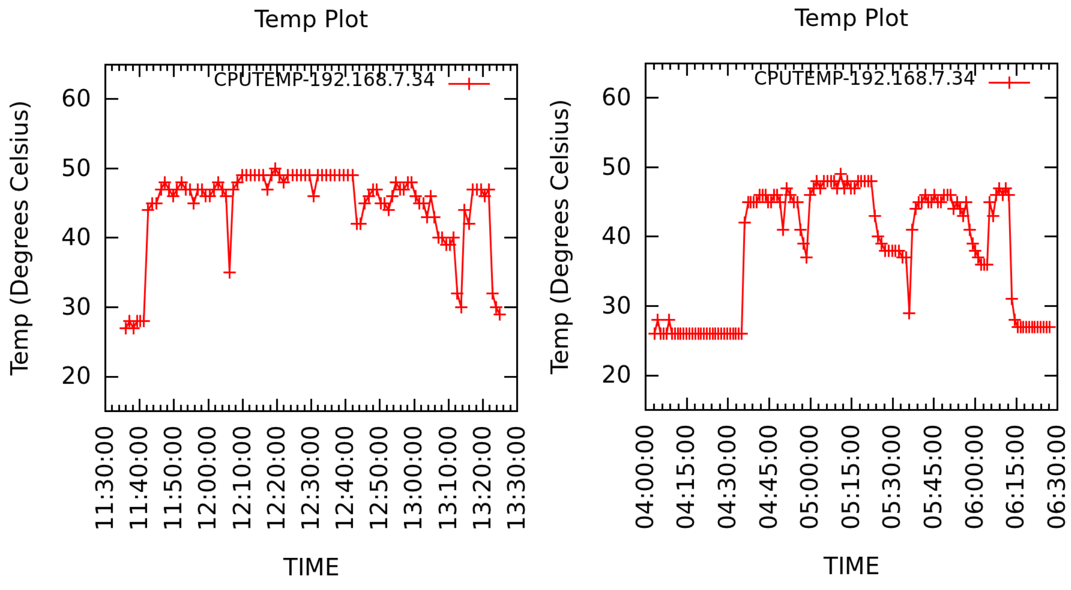


Figure 5.12: Temperature of Server 34 classified as "MEDIUM" with FCFS(left) and TAS (right)

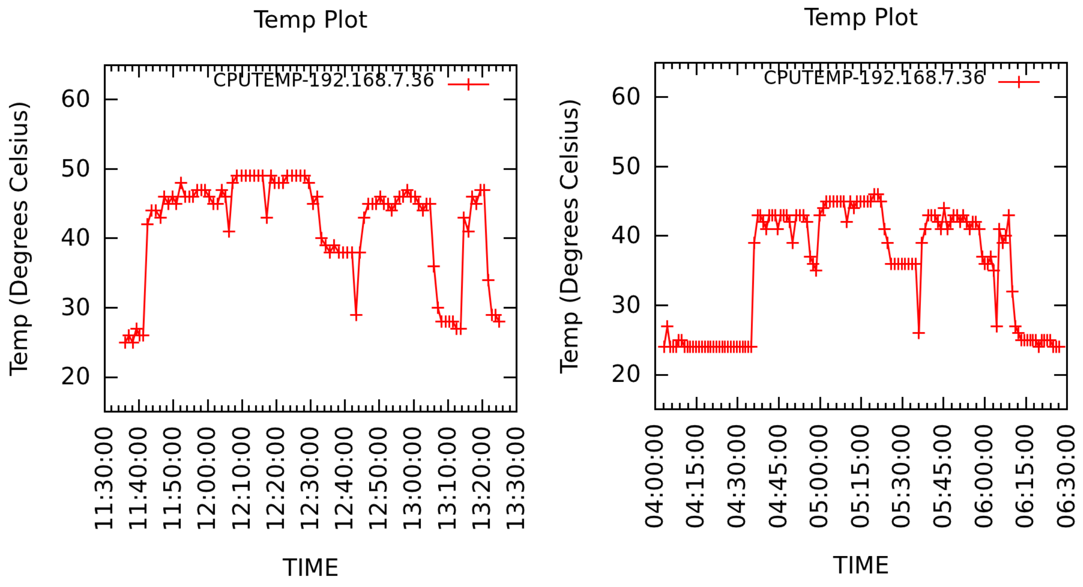
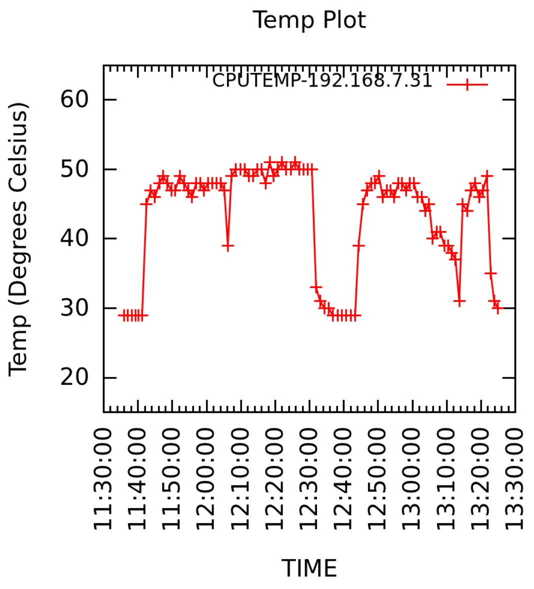


Figure 5.13: Temperature of Server 36 classified as "MEDIUM" with FCFS(left) and TAS (right)

5.6.3. Temperature profile of "MEDIUM" servers with "Large" delay

Figures 5.10,5.11,5.12 and 5.13 shows the corresponding temperature graphs when a server classified as "MEDIUM" is run with the FCFS scheduler and the TAS. For Server 9,10,34,36 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler. We see that the delay period gets over at around 4:40 after which the servers start executing their tasks.



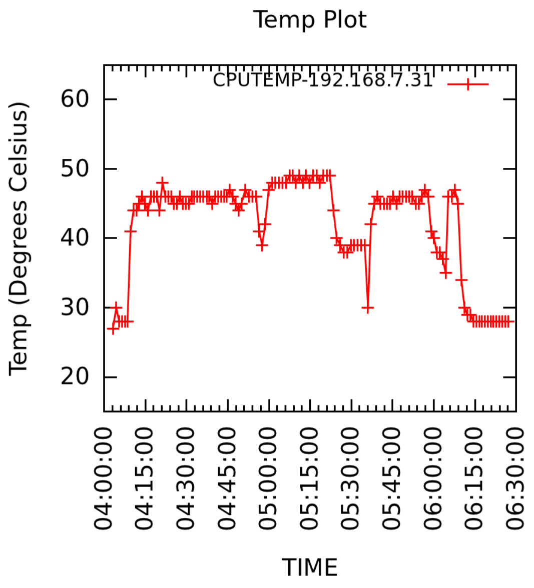
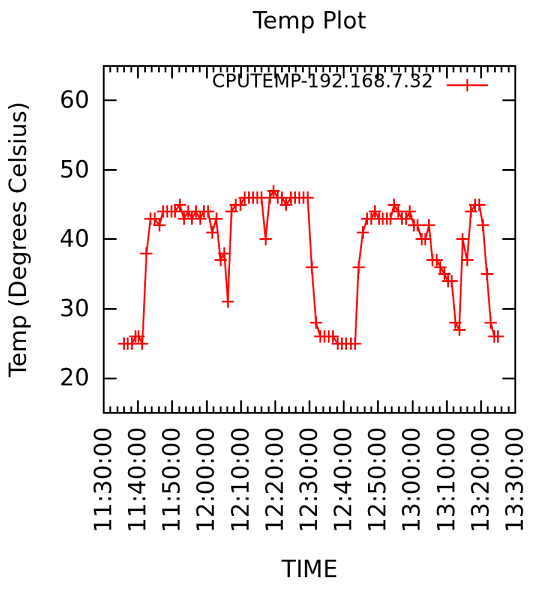


Figure 5.14: Temperature of Server 31 classified as "COLD" with FCFS(left) and TAS (right)

5.6.4. Temperature profile of "COLD" servers with "Large" delay

Figures 5.14,5.15 and 5.16 shows the corresponding temperature graphs when a server classified as "COLD" is run with the FCFS scheduler and the TAS. For Server 31,32,33 the graphs show that they run hotter when run with TAS than with the native FCFS scheduler. This is because it gets more tasks to execute because of the



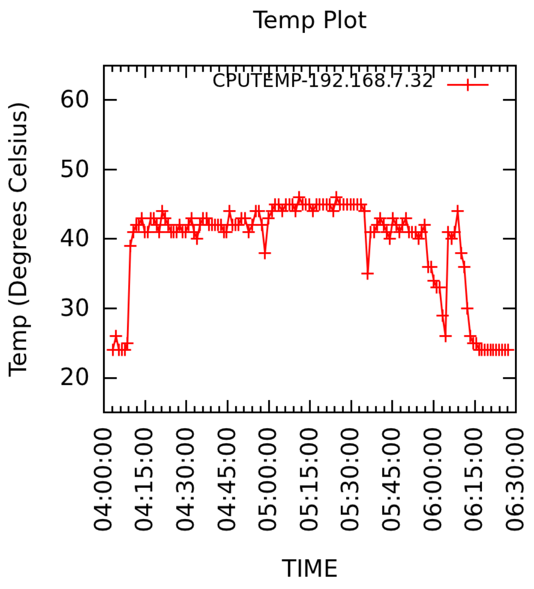
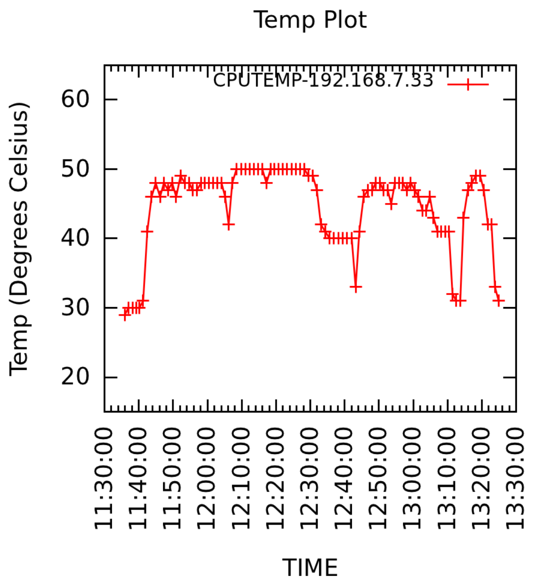


Figure 5.15: Temperature of Server 32 classified as "COLD" with FCFS(left) and TAS (right)



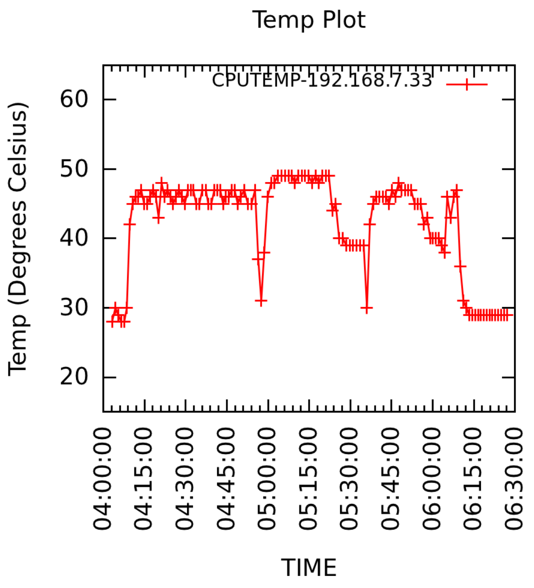
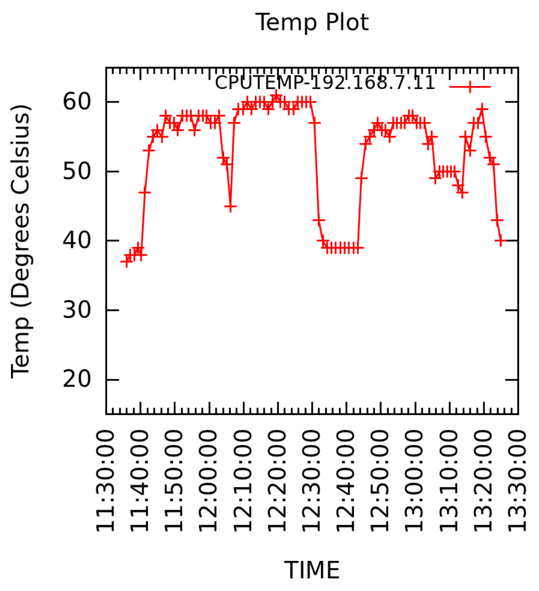


Figure 5.16: Temperature of Server 33 classified as "COLD" with FCFS(left) and TAS (right)

delay that is introduced for the "MEDIUM" and "HOT" servers consequently the higher temperatures.



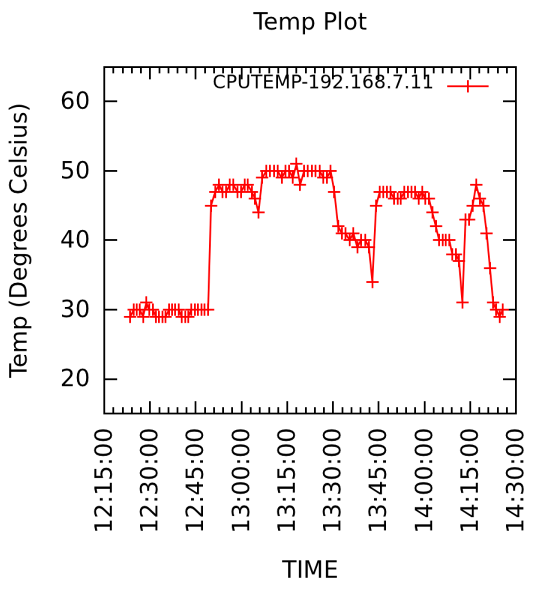


Figure 5.17: Temperature of Server 11 classified as "HOT" with FCFS(left) and TAS (right)

5.6.5. Temperature profile of "HOT" servers with "Small" delay

Figures 5.17,5.18,5.19 shows the corresponding temperature graphs when a server classified as HOT is run with the FCFS scheduler and the TAS. For Server 11,12,35 the graphs show that they run much cooler when run with TAS than with the native FCFS scheduler. Server 22 acts as the master and From Figure 5.20 we see that around 12:50, the temperature is around 43 but then shoots up to 52 once it starts executing the tasks.

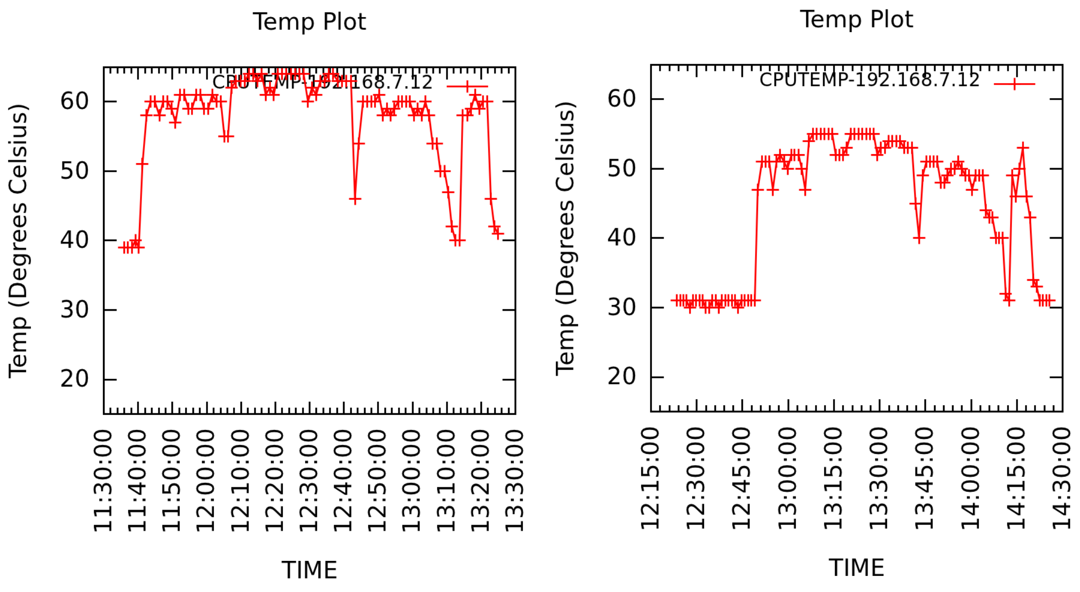


Figure 5.18: Temperature of Server 12 classified as “HOT” with FCFS(left) and TAS (right)

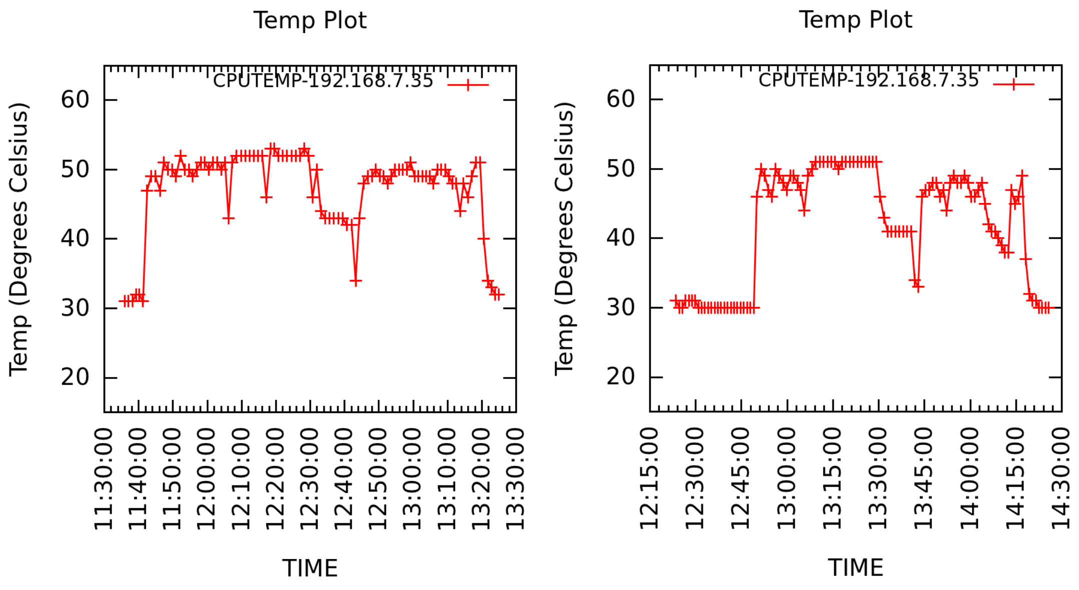


Figure 5.19: Temperature of Server 35 classified as “HOT” with FCFS(left) and TAS (right)

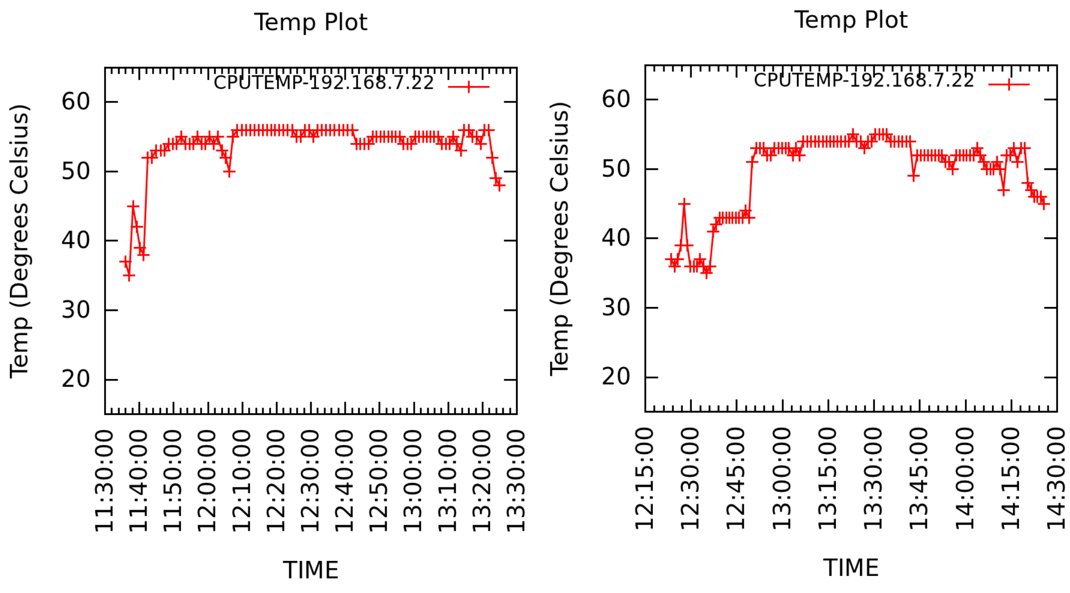


Figure 5.20: Temperature of Server 22 classified as "HOT" with FCFS(left) and TAS (right)

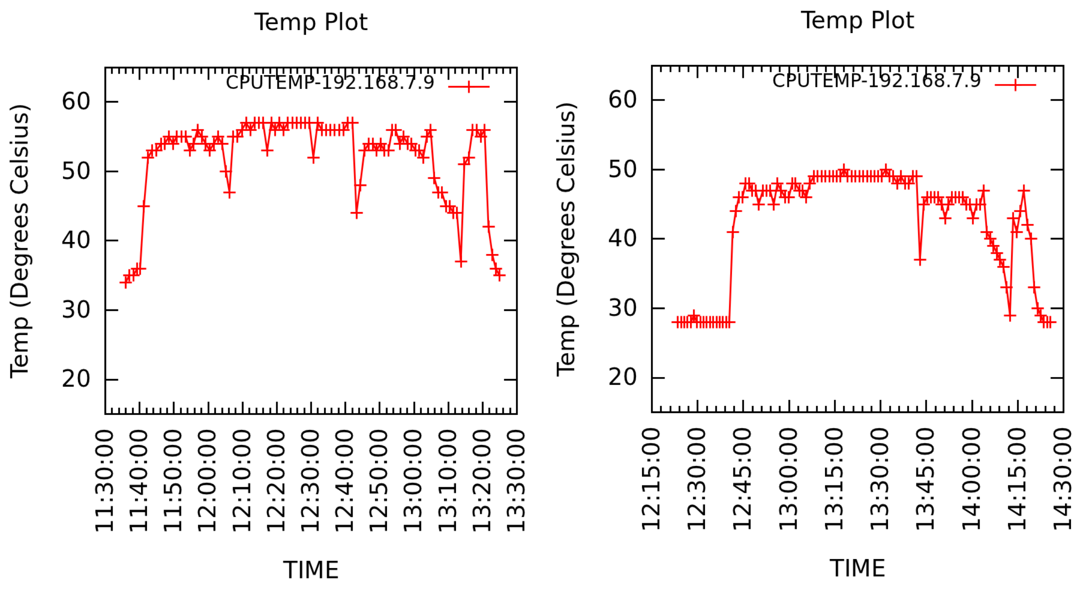


Figure 5.21: Temperature of Server 9 classified as "MEDIUM" with FCFS(left) and TAS (right)

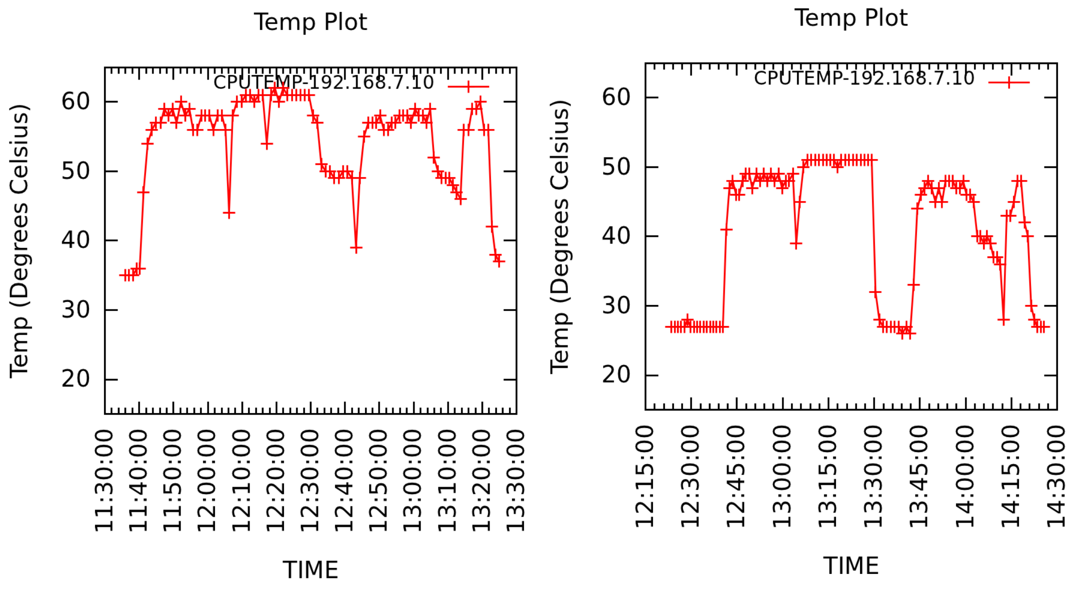


Figure 5.22: Temperature of Server 10 classified as "MEDIUM" with FCFS(left) and TAS (right)

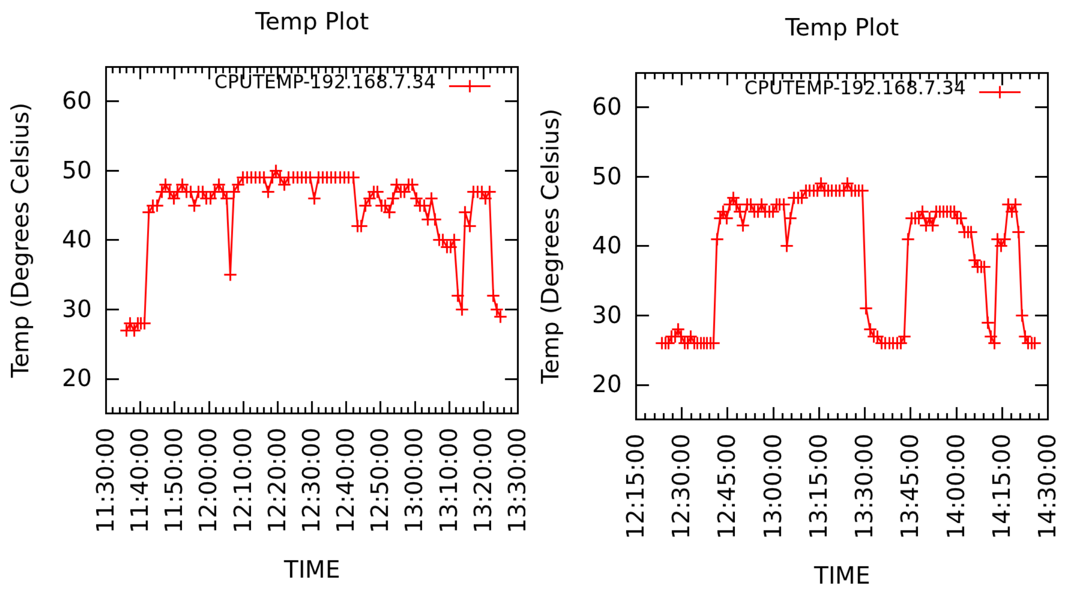
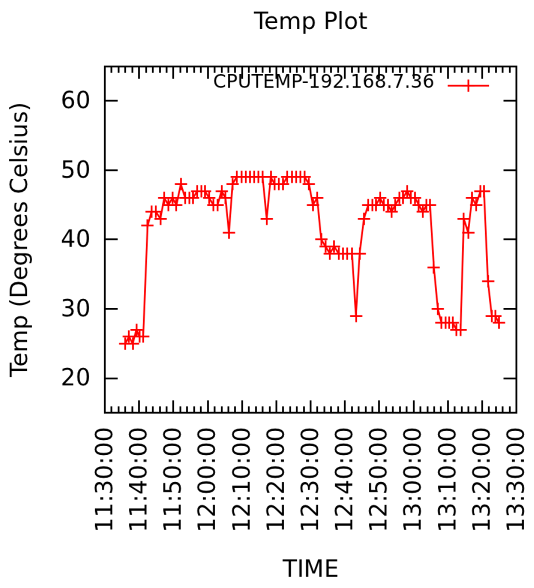


Figure 5.23: Temperature of Server 34 classified as "MEDIUM" with FCFS(left) and TAS (right)



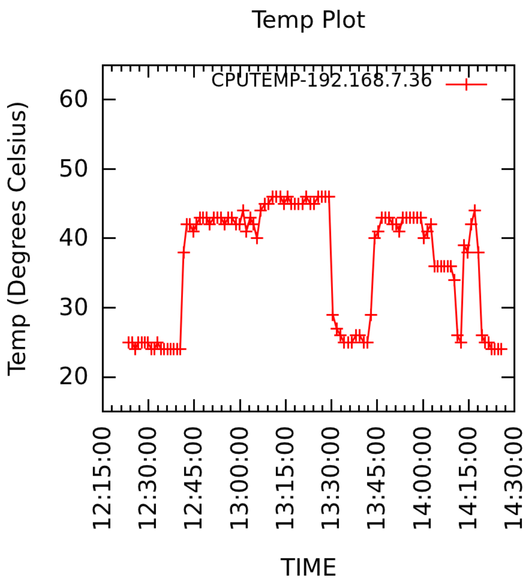


Figure 5.24: Temperature of Server 36 classified as "MEDIUM" with FCFS(left) and TAS (right)

Temperature profile of "MEDIUM" servers with "Small" delay

Figures 5.21,5.22,5.23 and 5.24 shows the corresponding temperature graphs when a server classified as MEDIUM is run with the FCFS scheduler and the TAS. For Server 9,10,34,36 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler. We see that the delay period gets over at around 12:38 after which the servers start executing their tasks as a result the temperature increases from 28 to 48.

5.6.6. Temperature profile of "COLD" servers with "Small" delay

Figures 5.25, 5.26 and 5.27 shows the corresponding temperature graphs when a server classified as "COLD" is run with the FCFS scheduler and the TAS. For Server 31,32,33 the graphs show that they run hotter when run with TAS than with the

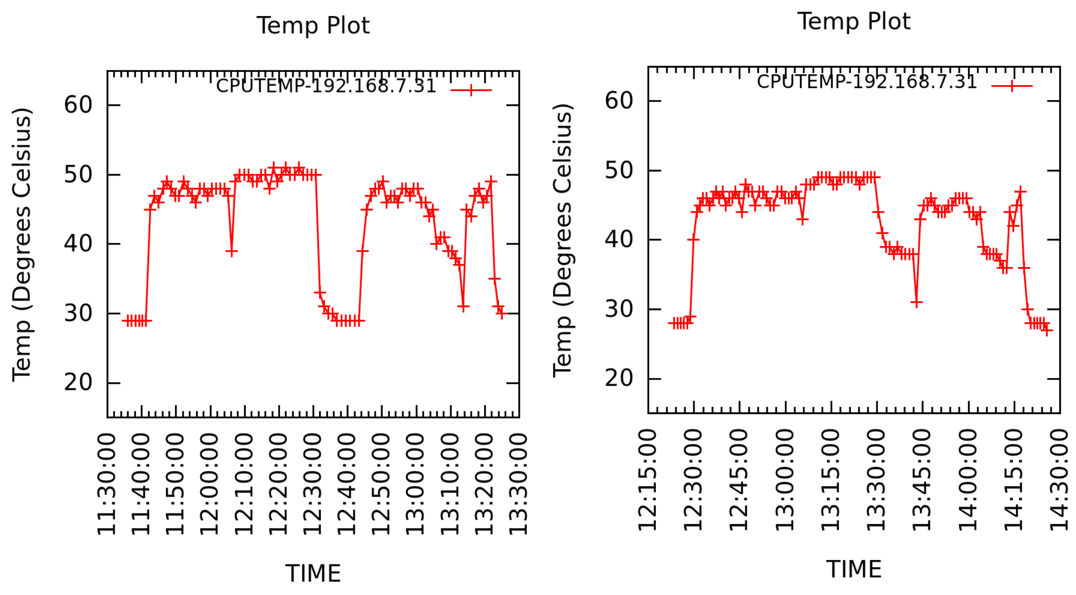


Figure 5.25: Temperature of Server 31 classified as "COLD" with FCFS(left) and TAS (right)

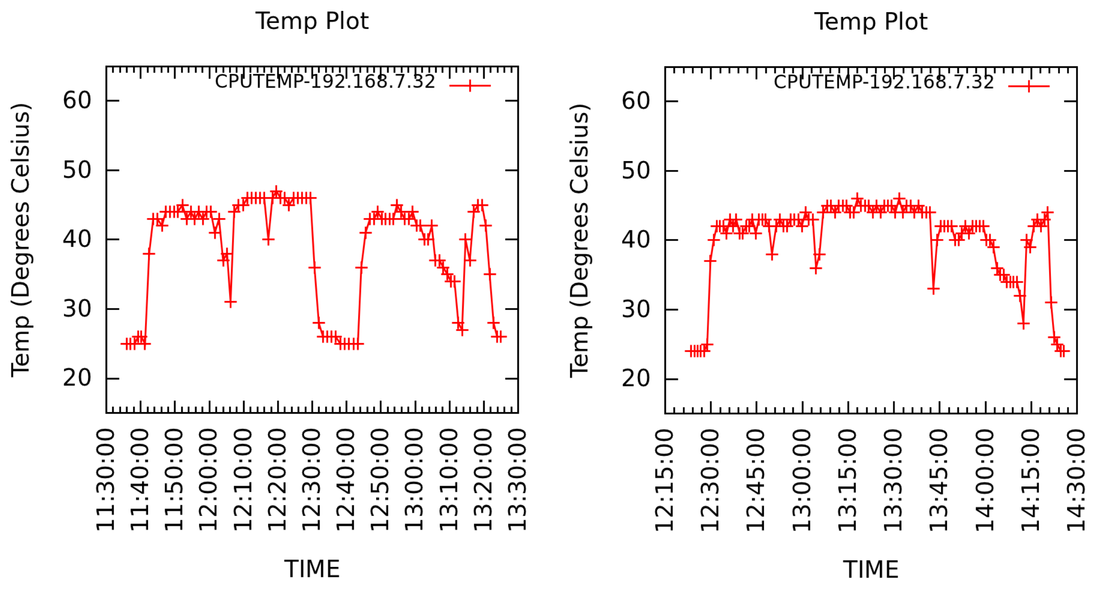


Figure 5.26: Temperature of Server 32 classified as "COLD" with FCFS(left) and TAS (right)

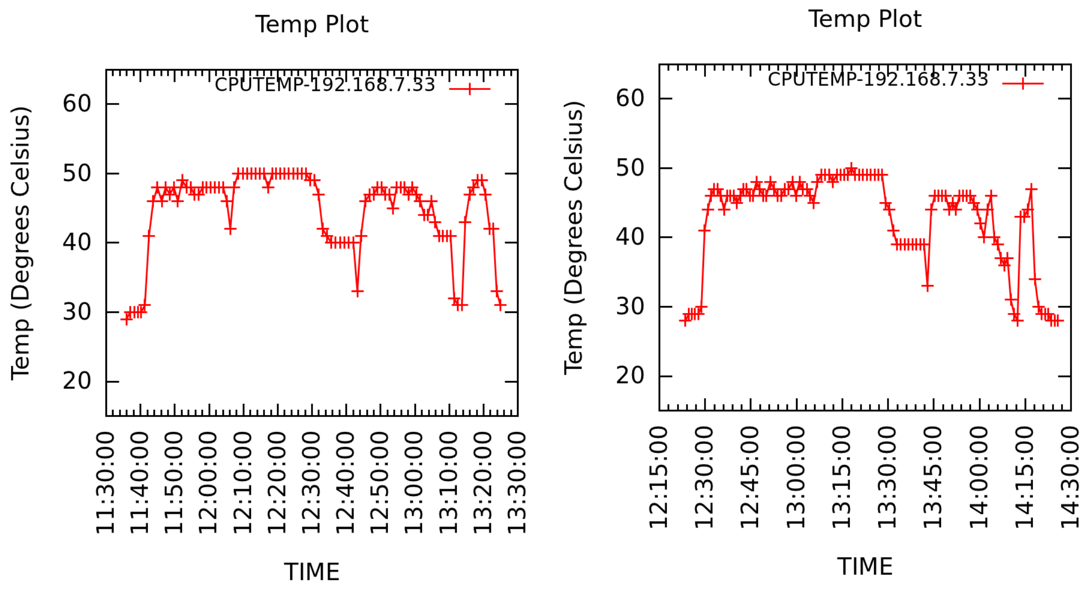


Figure 5.27: Temperature of Server 33 classified as "COLD" with FCFS(left) and TAS (right)

native FCFS scheduler. This is because it gets more tasks to execute because of the delay that is introduced for the "MEDIUM" and "HOT" servers.

5.7. TAS VERSUS FCFS FOR I/O INTENSIVE WORKLOAD (TERASORT) ON HADOOP

In this experiment we evaluate TAS versus FCFS when using TeraSort benchmark which is an I/O intensive workload. We run TeraSort three times using FCFS, and TAS with two delay interval of large and small as described below.

Table 5.6 shows the initial temperatures of the servers based on which they have been classified as either "HOT", "COLD" or "MEDIUM" as well as the "Large" delay interval based on the number of requests. For the "HOT" servers this delay period was set to 0.8 times running time for the job while for the "MEDIUM" servers, it was set to 0.5 times the running time of the job in FCFS.

Table 5.6: Classification of Servers based on initial temperature with Large Delay for TeraSort workload

Server	Type	Initial Temperature (C)	Classification	Delay (requests)
9	Slave	37	MEDIUM	200
10	Slave	38	MEDIUM	200
11	Slave	43	HOT	350
12	Slave	43	HOT	350
22	Master	40	HOT	350
31	Slave	29	COLD	0
32	Slave	27	COLD	0
33	Slave	29	COLD	0
34	Slave	32	MEDIUM	200
35	Slave	39	HOT	350
36	Slave	35	MEDIUM	200

Table 5.7 shows the initial temperatures of the servers based on which they have been classified as either “HOT”, “COLD” or “MEDIUM” and the “Small” delay interval for each of the classification types. The servers classified as “HOT” were given delay interval of 0.5 times the running time of the job while for the “MEDIUM” servers the delay interval was 0.25 times the running time of the job

Table 5.7: Classification of Servers based on initial temperature with Small delay for TeraSort workload

Server	Type	Initial Temperature (C)	Classification	Delay (requests)
9	Slave	37	MEDIUM	100
10	Slave	38	MEDIUM	100
11	Slave	43	HOT	200
12	Slave	43	HOT	200
22	Master	40	HOT	200
31	Slave	29	COLD	0
32	Slave	27	COLD	0
33	Slave	29	COLD	0
34	Slave	32	MEDIUM	100
35	Slave	39	HOT	200
36	Slave	35	MEDIUM	100

We run TeraSort three times using TAS according to the configuration given Table 5.6 and Table 5.7, as well as when using FCFS. Figure 5.28 summarizes the reduction in peak temperature using TAS compared with FCFS scheduler. We see that the temperatures are much less than the PageRank workload because it was more CPU intensive (compare Figure 5.5 with Figure 5.28). Since the temperature variation is less the reduction in temperature with TAS over FCFS is also considerably less for TeraSort when compared with PageRank workload.

For the “HOT” and “MEDIUM” servers, reduction in average temperature is comparatively greater for larger delay interval for the TAS scheduler and is much greater than the FCFS scheduler. For the “COLD” servers the average temperature is very similar for the FCFS as well as the TAS scheduler.

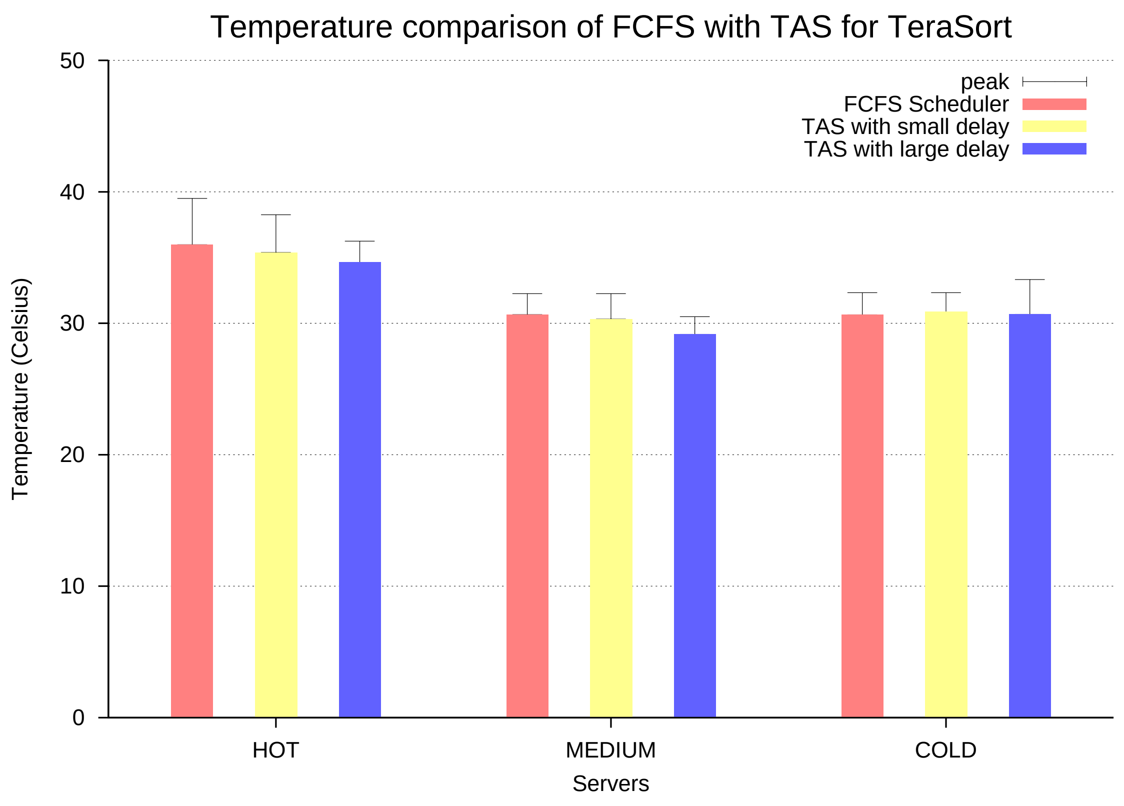


Figure 5.28: Peak Temperature of FCFS compared with TAS with large and small delay

5.7.1. Temperature profile of “HOT” servers with “Large” delay

Figures 5.29, 5.30, 5.31 shows the corresponding temperature graphs when a server classified as “HOT” is run with the FCFS scheduler and the TAS on TeraSort workload. For Server 11,12,35 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler. Server 22 acts as the master and it also runs comparatively cooler with TAS than with the FCFS one as seen from Figure 5.32. The delay period exceeds the runtime of the task as result of which we see that there is no spike in temperatures as they do not execute any task

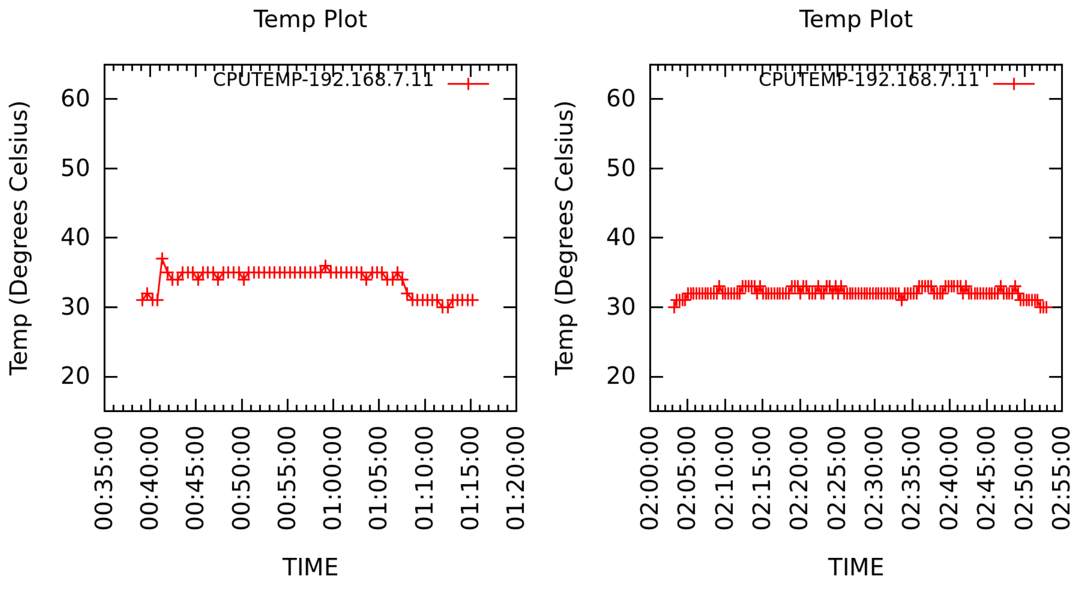


Figure 5.29: Temperature of Server 11 classified as "HOT" with FCFS(left) and TAS (right)

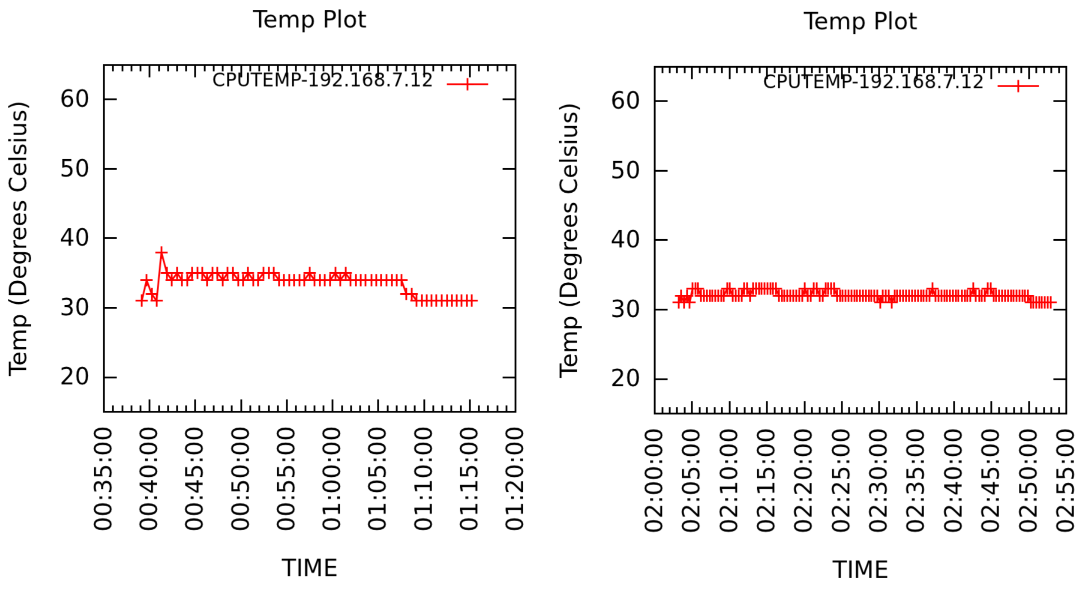


Figure 5.30: Temperature of Server 12 classified as "HOT" with FCFS(left) and TAS (right)

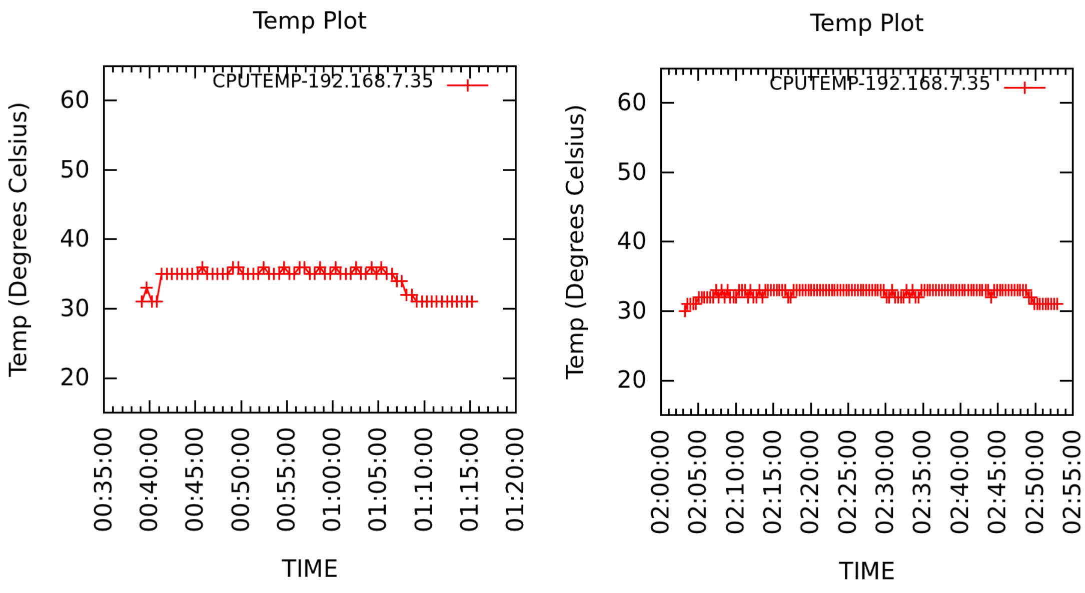


Figure 5.31: Temperature of Server 35 classified as "HOT" with FCFS(left) and TAS (right)

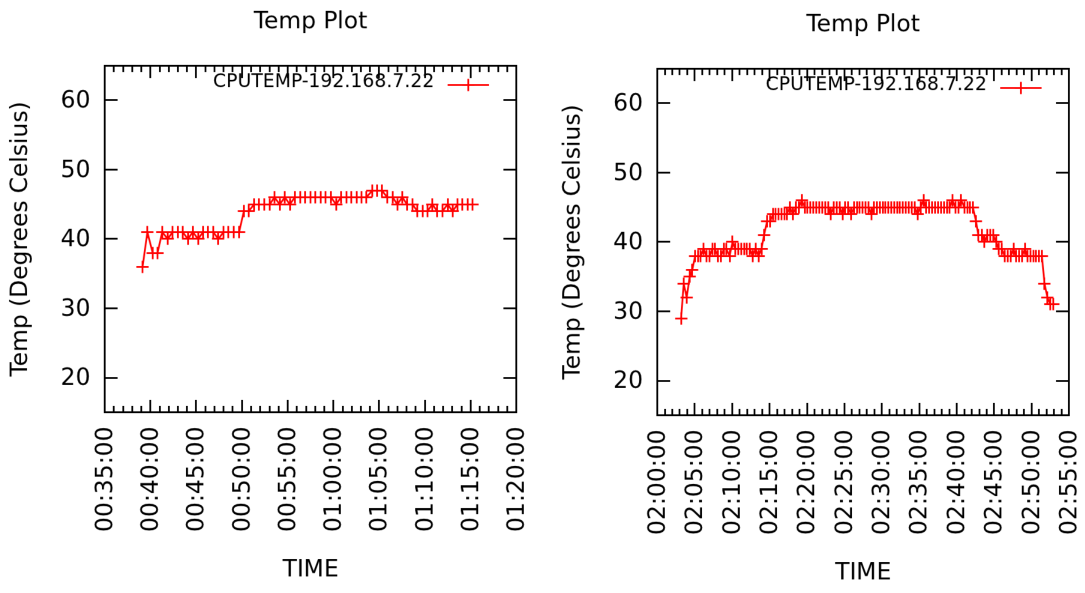


Figure 5.32: Temperature of Server 22 classified as "HOT" with FCFS(left) and TAS (right)

5.7.2. Temperature profile of “MEDIUM” servers with “Large” delay

Figures 5.33,5.34,5.35 and 5.36 shows the corresponding temperature graphs when a server classified as “MEDIUM” is run with the FCFS scheduler and the TAS. For Server 9,10,34,36 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler. We see that the delay period noticeable by a spike in temperature gets over at around 2:35 after which the servers start executing their tasks.

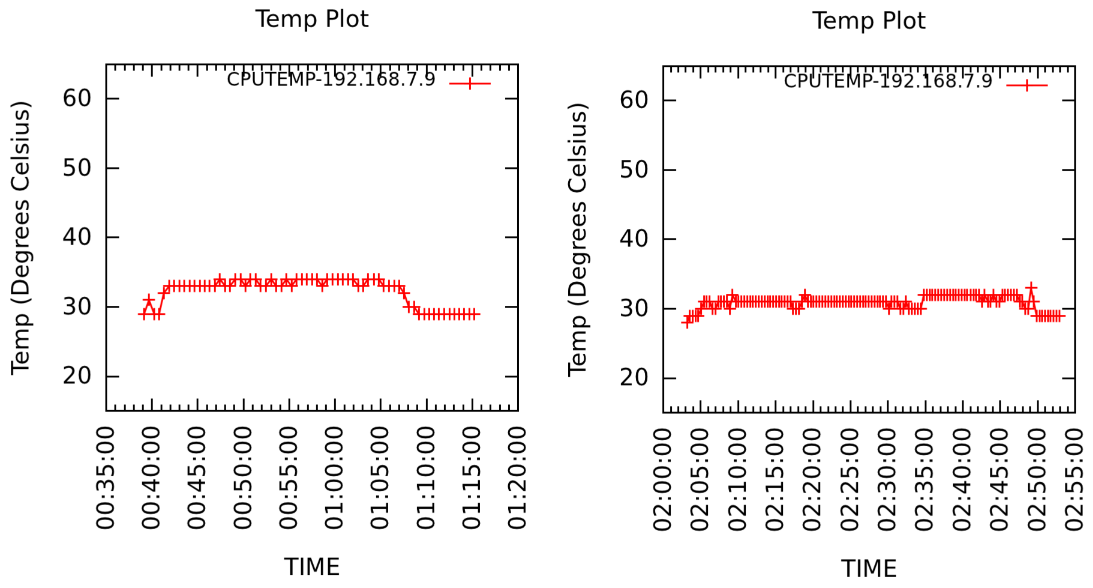


Figure 5.33: Temperature of Server 9 classified as “MEDIUM” with FCFS(left) and TAS (right)

5.7.3. Temperature profile of “COLD” servers with “Large” delay

Figures 5.37,5.38 and 5.39 shows the corresponding temperature graphs when a server classified as “COLD” is run with the FCFS scheduler and the TAS. For Server 31,32,33 the graphs show that they run hotter when run with TAS than with the

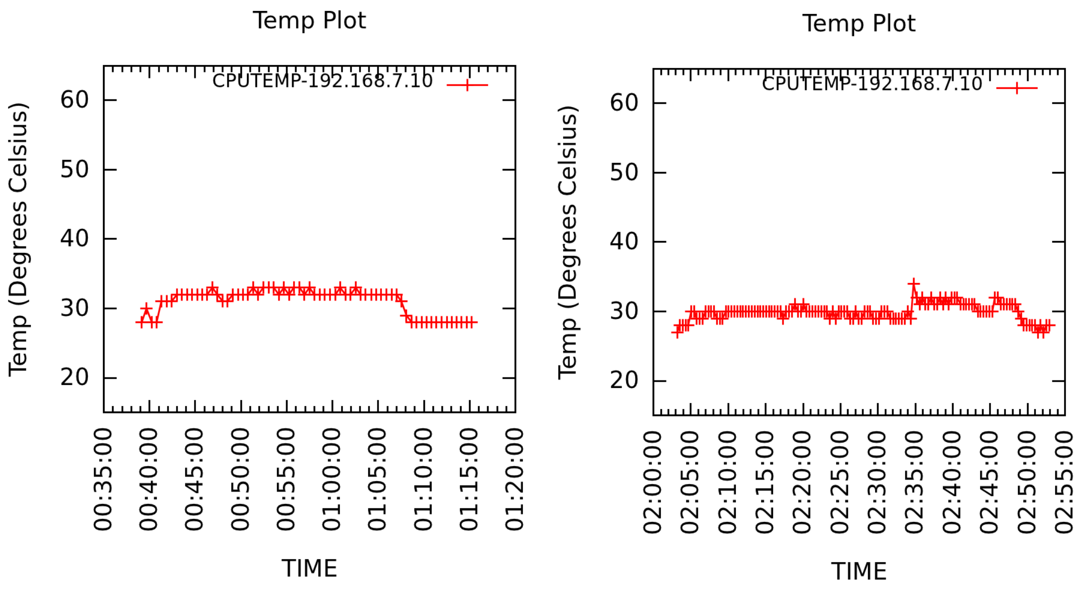


Figure 5.34: Temperature of Server 10 classified as "MEDIUM" with FCFS(left) and TAS (right)

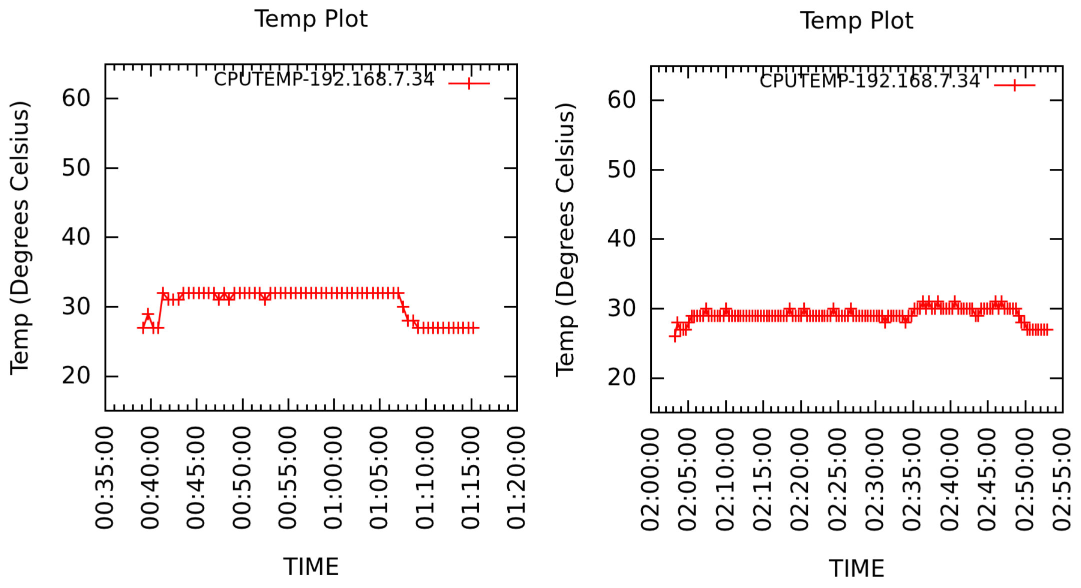
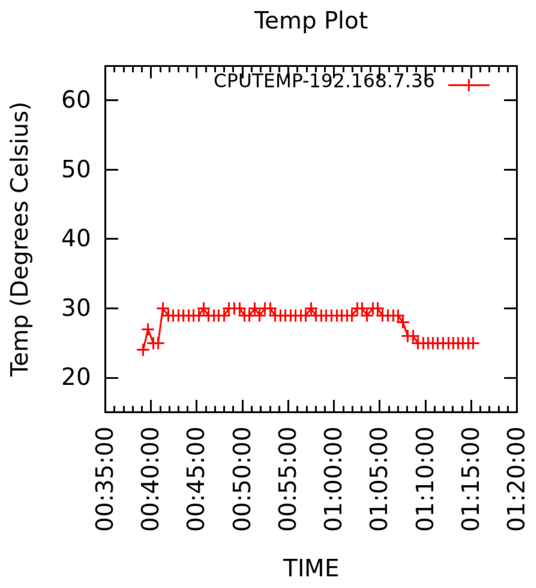


Figure 5.35: Temperature of Server 34 classified as "MEDIUM" with FCFS(left) and TAS (right)



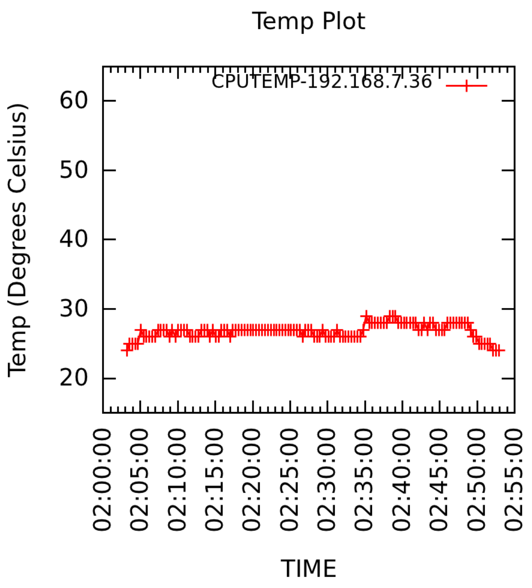


Figure 5.36: Temperature of Server 36 classified as "MEDIUM" with FCFS(left) and TAS (right)

native FCFS scheduler. This is because it gets more tasks to execute because of the delay that is introduced for the "MEDIUM" and "HOT" servers consequently the higher temperatures. We see that there is bit of drop in the temperature at around 2:35 when the delay period for the "MEDIUM" servers gets over and they are able to execute their tasks

5.7.4. Temperature profile of "HOT" servers with "Small" delay

Figures 5.40,5.41,5.42 shows the corresponding temperature graphs when a server classified as "HOT" is run with the FCFS scheduler and the TAS on TeraSort workload. For Server 11,12,35 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler before 9:45. The delay period ends at 9:45 after which it starts executing the tasks hence a spike in temperature. Server 22 which is the master runs noticeably cooler in TAS as compared to the FCFS scheduler as

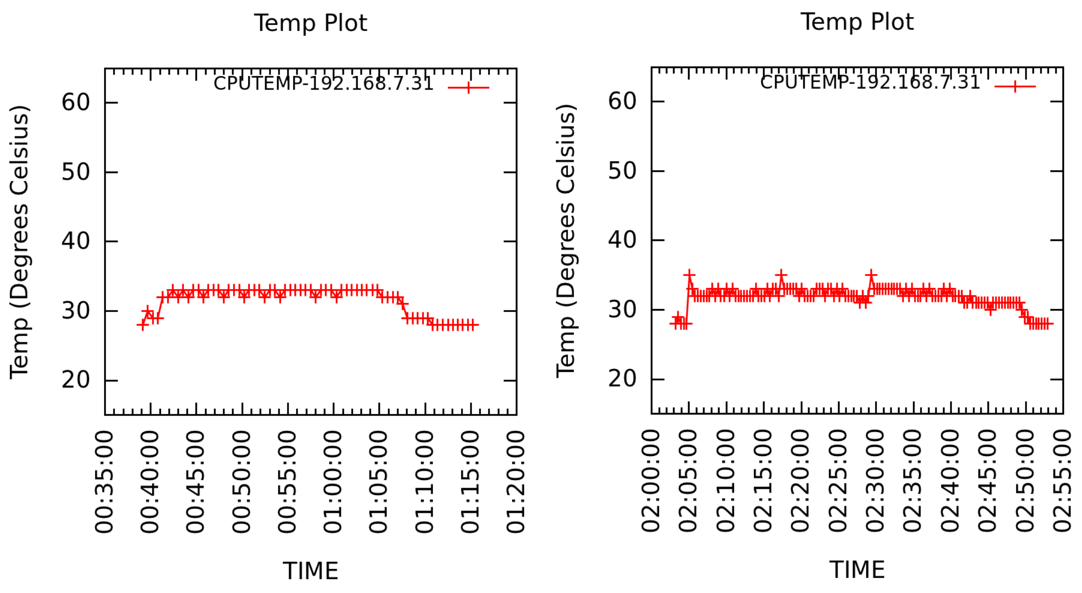


Figure 5.37: Temperature of Server 31 classified as "COLD" with FCFS(left) and TAS (right)

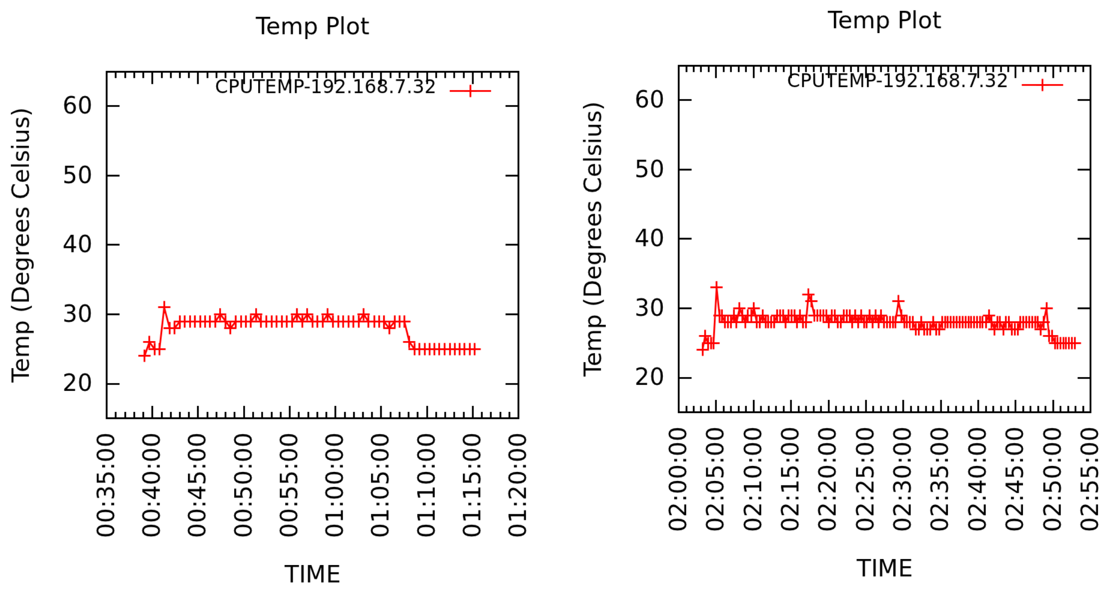


Figure 5.38: Temperature of Server 32 classified as "COLD" with FCFS(left) and TAS (right)

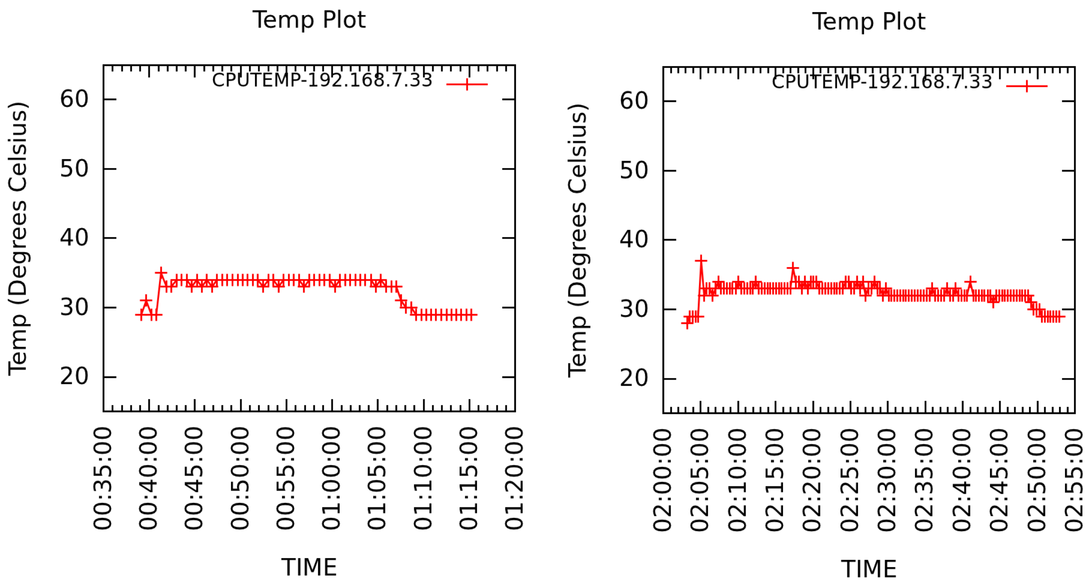


Figure 5.39: Temperature of Server 33 classified as "COLD" with FCFS(left) and TAS (right)

shown in figure 5.43

5.7.5. Temperature profile of "MEDIUM" servers with "Small" delay

Figures 5.44,5.45,5.46 and 5.47 shows the corresponding temperature graphs when a server classified as "MEDIUM" is run with the FCFS scheduler and the TAS. For Server 9,10,34,36 the graphs show that they run cooler when run with TAS than with the native FCFS scheduler till about 9:39 after which it starts executing the tasks and its temperature mirrors that of the FCFS scheduler.

5.7.6. Temperature profile of "COLD" servers with "Small" delay

Figures 5.48,5.49 and 5.50 shows the corresponding temperature graphs when a server classified as "COLD" is run with the FCFS scheduler and the TAS. For Server 31,32,33 the graphs show that they run hotter when run with TAS than with the

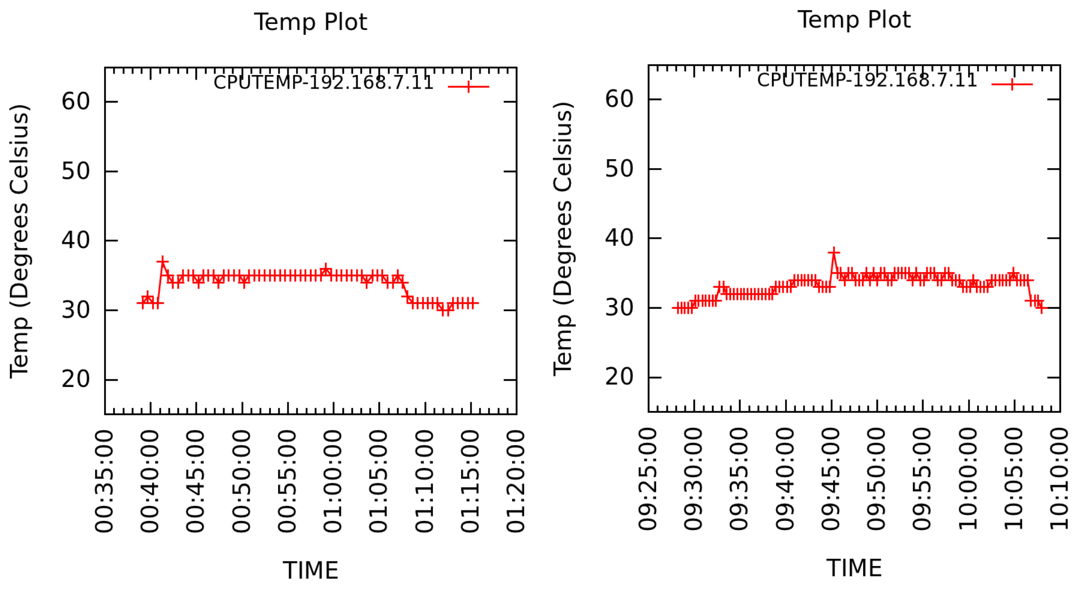


Figure 5.40: Temperature of Server 11 classified as "HOT" with FCFS(left) and TAS (right)

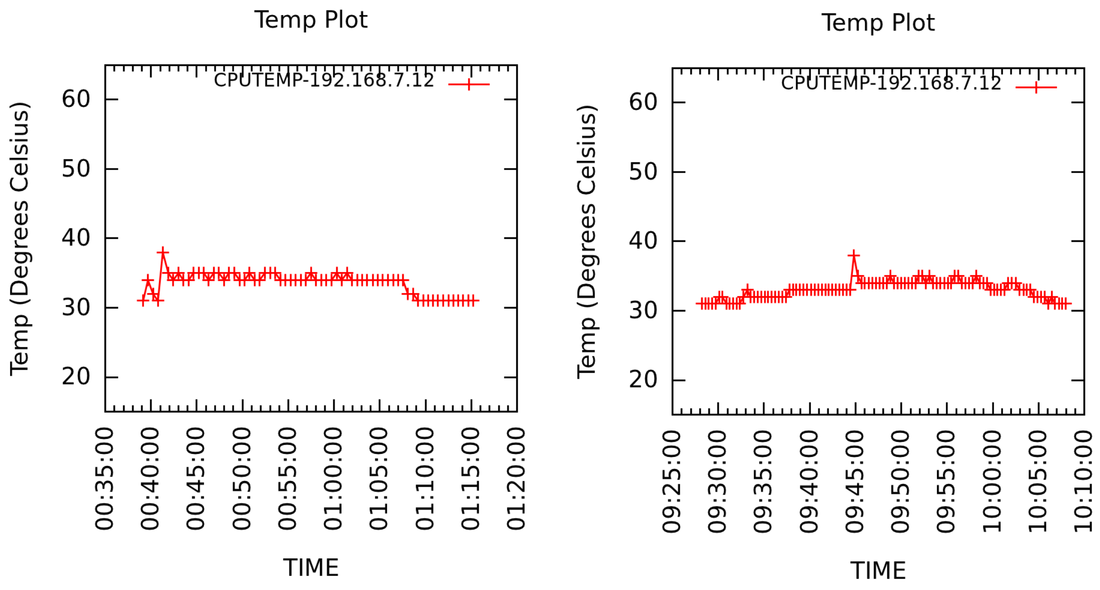


Figure 5.41: Temperature of Server 12 classified as "HOT" with FCFS(left) and TAS (right)

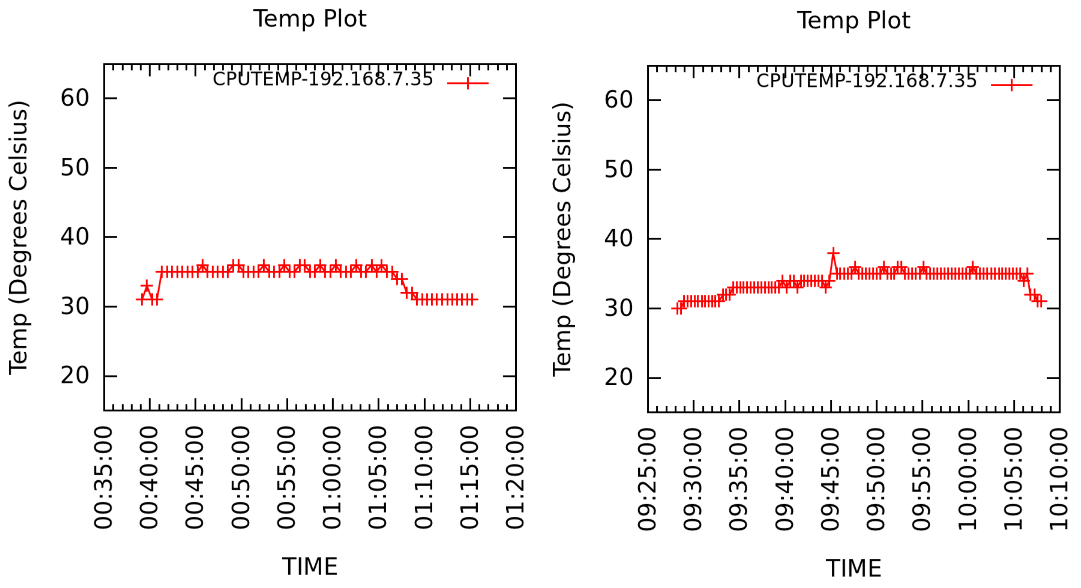


Figure 5.42: Temperature of Server 35 classified as "HOT" with FCFS(left) and TAS (right)

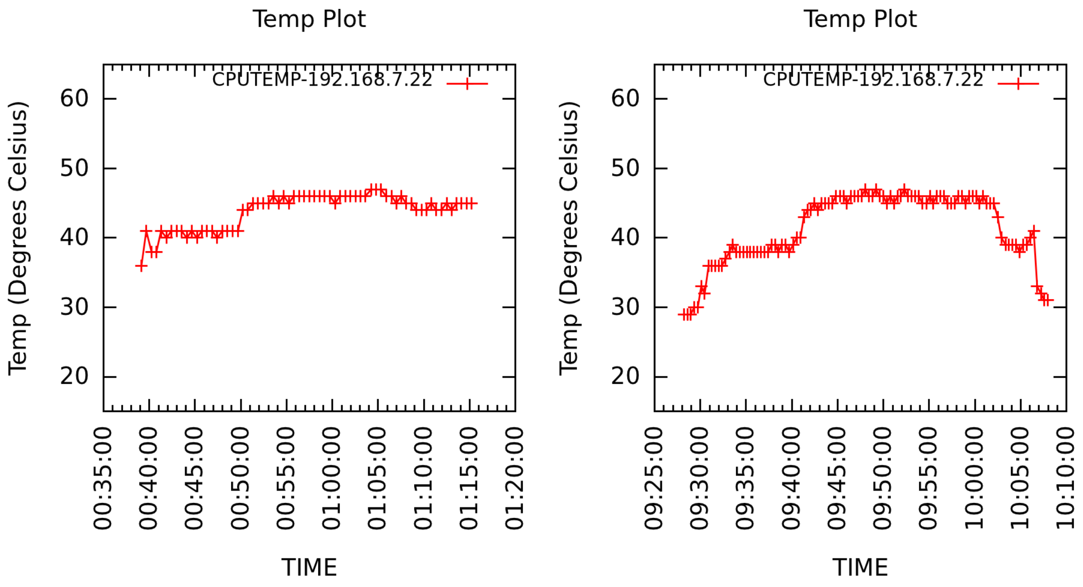


Figure 5.43: Temperature of Server 22 classified as "HOT" with FCFS(left) and TAS (right)

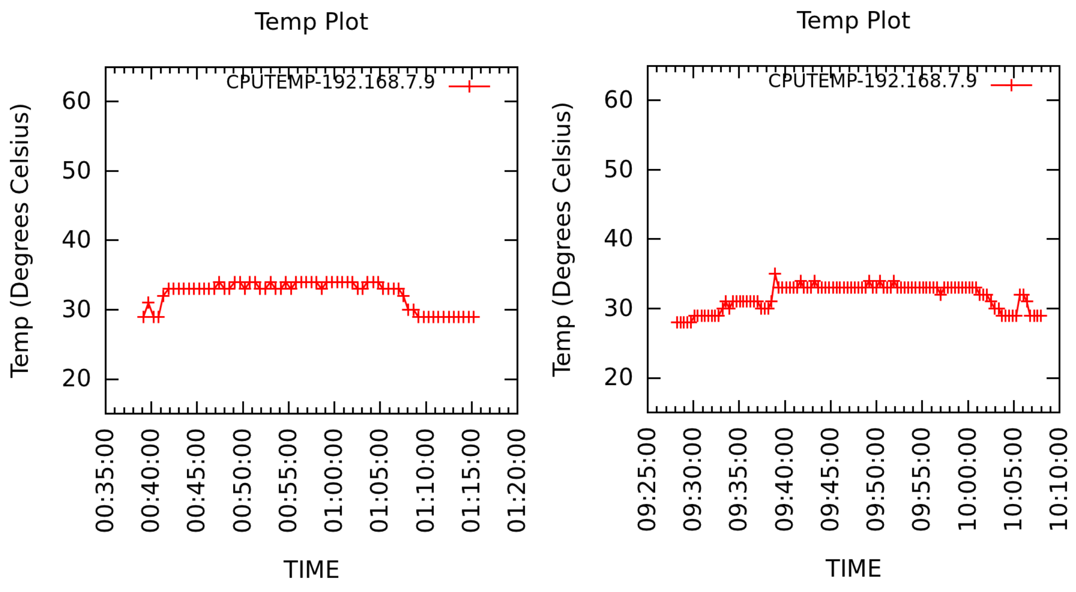


Figure 5.44: Temperature of Server 9 classified as "MEDIUM" with FCFS(left) and TAS (right)

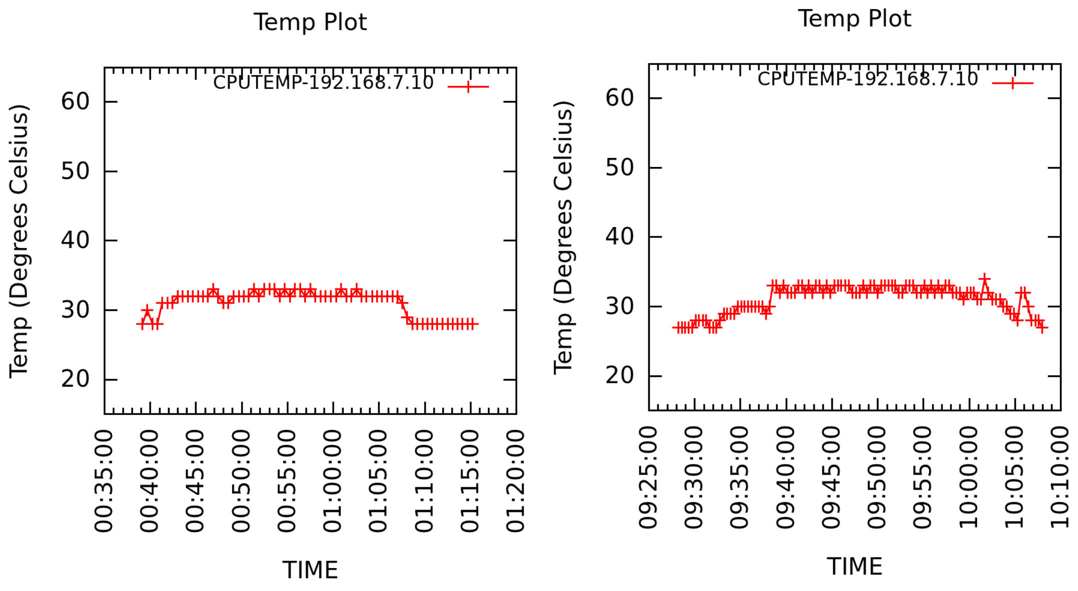


Figure 5.45: Temperature of Server 10 classified as "MEDIUM" with FCFS(left) and TAS (right)

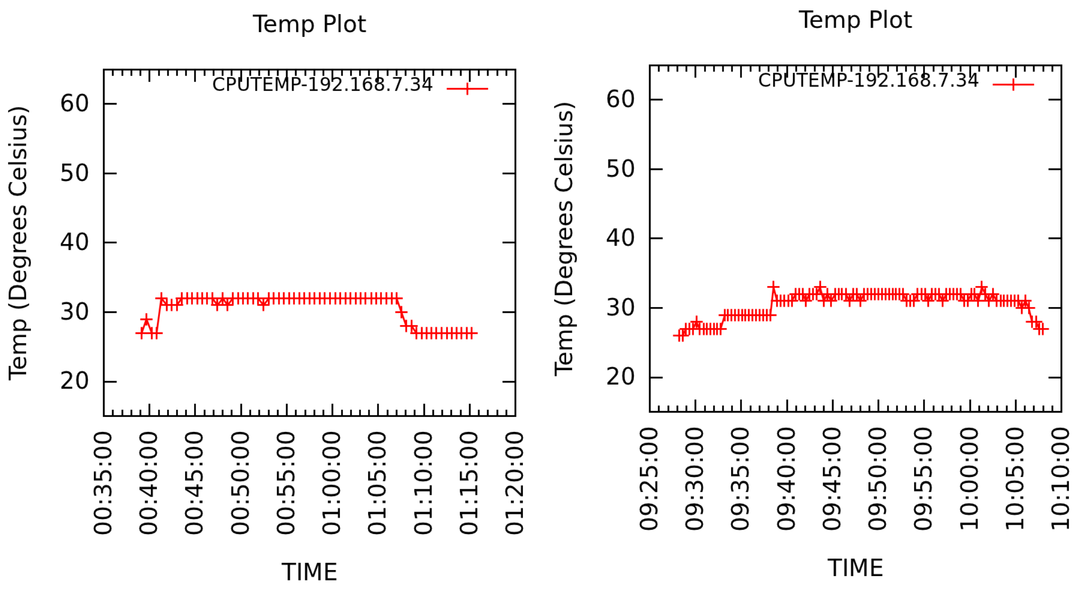


Figure 5.46: Temperature of Server 34 classified as "MEDIUM" with FCFS(left) and TAS (right)

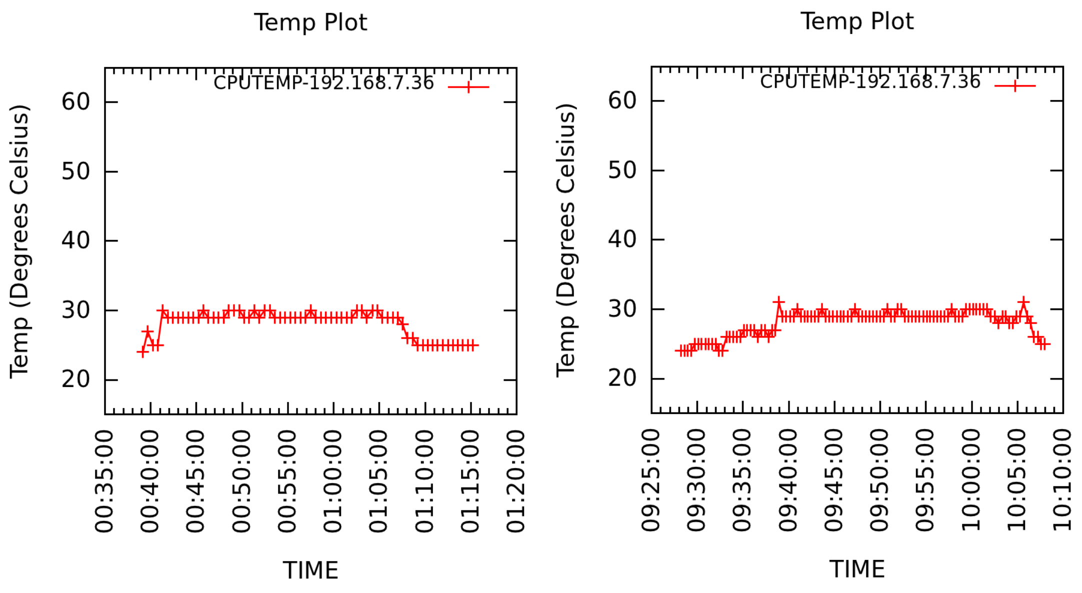
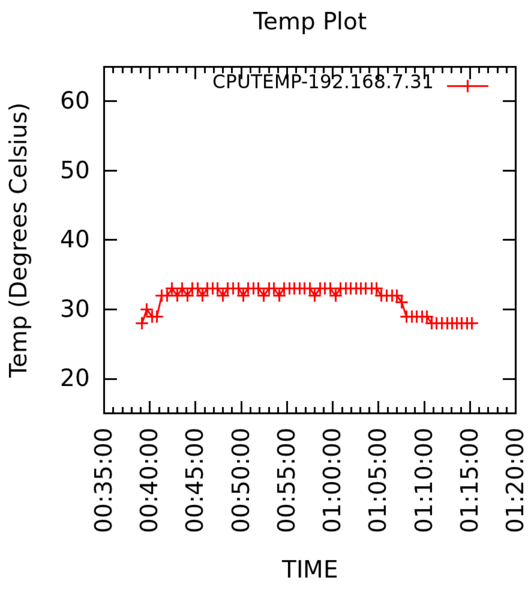


Figure 5.47: Temperature of Server 36 classified as "MEDIUM" with FCFS(left) and TAS (right)

native FCFS scheduler. This is because it gets more tasks to execute because of the delay that is introduced for the “MEDIUM” and “HOT” servers consequently the higher temperatures.



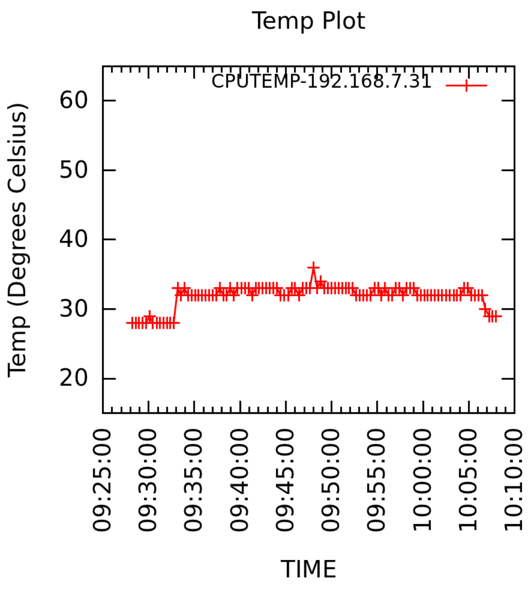


Figure 5.48: Temperature of Server 31 classified as “COLD” with FCFS(left) and TAS (right)

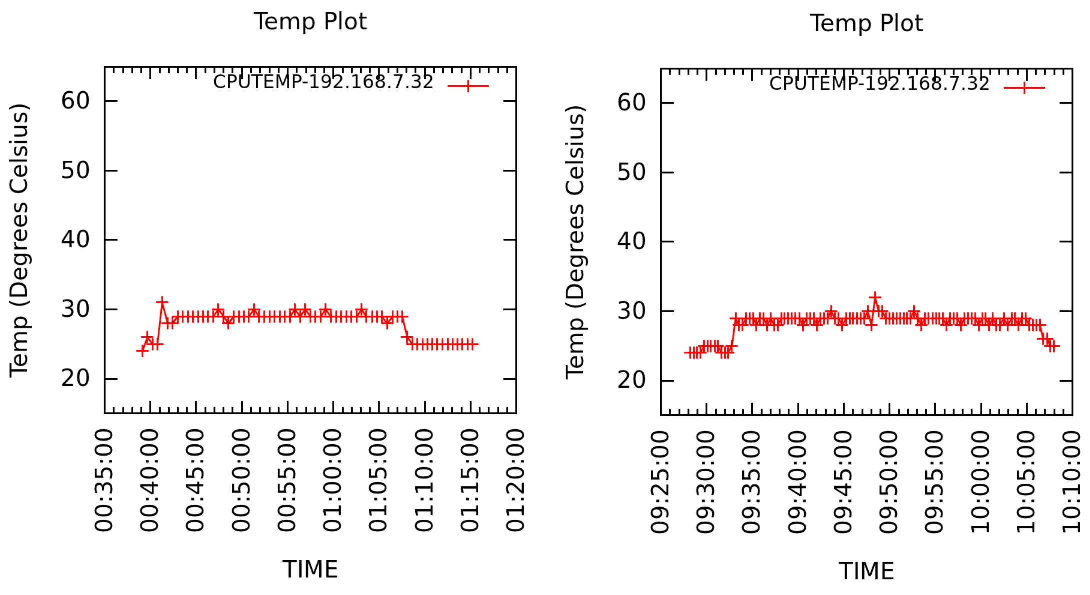


Figure 5.49: Temperature of Server 32 classified as "COLD" with FCFS(left) and TAS (right)

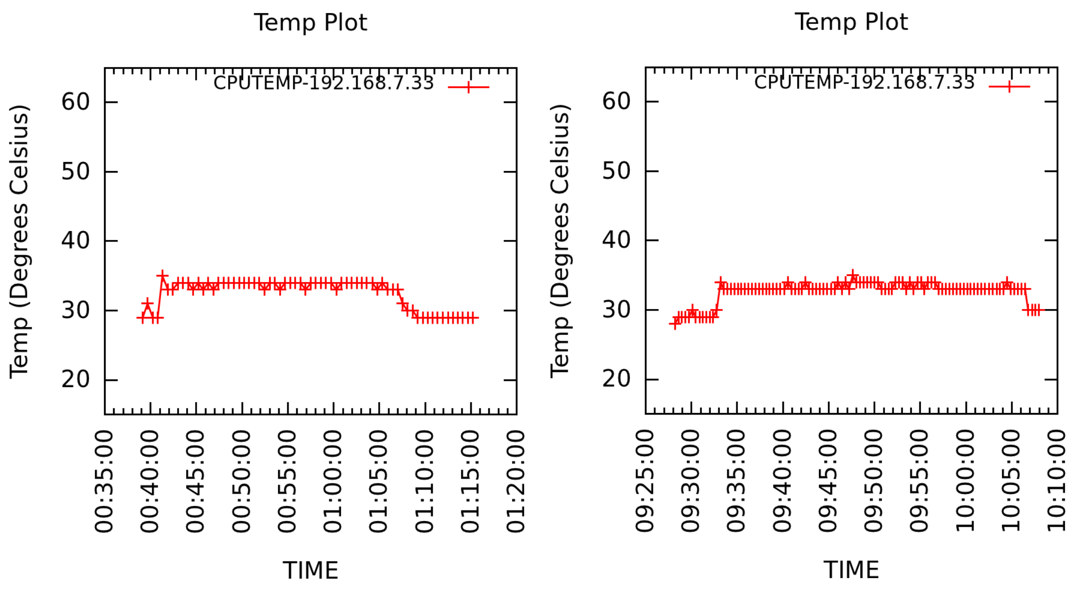


Figure 5.50: Temperature of Server 33 classified as "COLD" with FCFS(left) and TAS (right)

Chapter 6

CONCLUSION AND FUTURE WORK

6.1. CONCLUSION

We have proposed a Thermal Aware Scheduler (TAS) for Hadoop MapReduce framework which extends the First Come First Server (FCFS) scheduler to make it thermally aware. The Thermal Aware Scheduler distributes tasks preferentially to servers based on their classification as “HOT”, “MEDIUM” or “COLD”. We then tested our scheduler on two benchmarks, PageRank and Terasort and showed that preferential task distribution decreases the peak temperature as well as cooling power. The decrease in the peak temperature was more for PageRank than for TeraSort as Pagerank is more CPU intensive.

6.2. FUTURE WORK

Hadoop 1.0.3 has several limitations with regards to scalability with increased cluster size and jobs. The JobTracker was found to be inadequate to handle both management of resources as well as scheduling. The number of maps and reduce slots are static which does not fit well to varying workloads. Hadoop suits batch workload processing but not real time processing of workloads. The above limitations provide incentive to improve Hadoop.

The thermal aware scheduler schedules tasks based on the initial temperatures of the server CPU which is an example of static scheduling. Heat recirculation matrix is another parameter which could be used to perform thermal awareness in Hadoop.

Incorporating dynamic temperature monitoring in scheduling the tasks would result in enhanced savings as well a scheduler which is more agile to temperature fluctuations. However increased dynamic scheduling entails monitoring with greater frequency as well as the increased overhead of network communication.

REFERENCES

Abbasi, Z., G. Varsamopoulos and S. K. Gupta, “Tacoma: Server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality”, ACM Transactions on Architecture and Code Optimization (TACO) **9**, 2, 11 (2012).

Banerjee, A., T. Mukherjee, G. Varsamopoulos and S. K. Gupta, “Cooling-aware and thermal-aware workload placement for green hpc data centers”, in “Green Computing Conference, 2010 International”, pp. 245–256 (IEEE, 2010).

Belady, C., A. Rawson, J. Pfleuger and T. Cader, “Green grid data center power efficiency metrics: Pue and dcie”, the green grid pp. 1–9 (2008).

Goiri, Í., K. Le, T. D. Nguyen, J. Guitart, J. Torres and R. Bianchini, “Greenhadoop: leveraging green energy in data-processing frameworks”, in “Proceedings of the 7th ACM european conference on Computer Systems”, pp. 57–70 (ACM, 2012).

Huang, S., J. Huang, J. Dai, T. Xie and B. Huang, “The hibench benchmark suite: Characterization of the mapreduce-based data analysis”, in “Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on”, pp. 41–51 (IEEE, 2010).

Li, S., T. Abdelzaher and M. Yuan, “Tapa: Temperature aware power allocation in data center with map-reduce”, in “Green Computing Conference and Workshops (IGCC), 2011 International”, pp. 1–8 (IEEE, 2011).

Moore, J. D., J. S. Chase, P. Ranganathan and R. K. Sharma, “Making scheduling” cool”: Temperature-aware workload placement in data centers.”, in “USENIX annual technical conference, General Track”, pp. 61–75 (2005).

Sharma, R. K., C. E. Bash, C. D. Patel, R. J. Friedrich and J. S. Chase, “Balance of power: Dynamic thermal management for internet data centers”, Internet Computing, IEEE **9**, 1, 42–49 (2005).

Tang, Q., S. Gupta and G. Varsamopoulos, “Thermal-aware task scheduling for data centers through minimizing heat recirculation”, in “Cluster Computing, 2007 IEEE International Conference on”, pp. 129–138 (IEEE, 2007).

Tang, Q., S. K. Gupta and G. Varsamopoulos, “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach”, Parallel and Distributed Systems, IEEE Transactions on **19**, 11, 1458–1472 (2008).

Varsamopoulos, G., A. Banerjee and S. K. Gupta, “Energy efficiency of thermal-aware job scheduling algorithms under various cooling models”, in “Contemporary Computing”, pp. 568–580 (Springer, 2009).

White, T., *Hadoop: the definitive guide* (O’Reilly, 2012).

APPENDIX A

IMPLEMENTATION DETAILS OF HADOOP THERMAL AWARE SCHEDULER

A.1. JOBTRACKER PSEUDOCODE

```
procedure PROCESSHEARTBEAT(heartbeat)
    status = heartbeat.status
    responseid = heartbeat.responseid
    ic = heartbeat.initialcontact
    accept = heartbeat.acceptnewtasks
    delay = heartbeat.delay
    if delay>0 then
        delay = delay - 1
        acceptnewtasks = FALSE
    else
        acceptnewtasks = TRUE
    end if
    heartbeat.status = status
    heartbeat.responseid = responseid +1
    heartbeat.acceptnewtasks = accept
    heartbeat.delay = delay
return heartbeat
end procedure
```

A.2. TASKTRACKER PSEUDOCODE

```
procedure SETDELAY(temperature)
    if temperature>38 then
        delay = $d_{max}$
    end if
    if temperature<38 and temperature>30 then
        delay = $d_{med}$
    end if
    if temperature<30 then
        delay = 0
    end if
return delay
end procedure
```

A.3. MODIFIED CODE FOR JOBTRACKER.JAVA

```
// JobTracker.java

public synchronized HeartbeatResponse heartbeat(TaskTrackerStatus status,
                                                boolean restarted,
                                                boolean initialContact,
                                            boolean acceptNewTasks,
                                            short responseId,
                                            int delay)
throws IOException {
if (LOG.isDebugEnabled()) {
  LOG.debug("Got heartbeat from: " + status.getTrackerName() +
            " (restarted: " + restarted +
            " initialContact: " + initialContact +
            " acceptNewTasks: " + acceptNewTasks + ")" +
            " with responseId: " + responseId + " with delay " + delay);
}

// Make sure heartbeat is from a tasktracker allowed by the jobtracker.
if (!acceptTaskTracker(status)) {
  throw new DisallowedTaskTrackerException(status);
}

// First check if the last heartbeat response got through
String trackerName = status.getTrackerName();
long now = clock.getTime();
if (restarted) {
  faultyTrackers.markTrackerHealthy(status.getHost());
} else {
  faultyTrackers.checkTrackerFaultTimeout(status.getHost(), now);
}

HeartbeatResponse prevHeartbeatResponse =
  trackerToHeartbeatResponseMap.get(trackerName);
boolean addRestartInfo = false;
int prevdelay=delay;
if (initialContact != true) {
  // If this isn't the 'initial contact' from the tasktracker,
  // there is something seriously wrong if the JobTracker has
  // no record of the 'previous heartbeat'; if so, ask the
  // tasktracker to re-initialize itself.
  if (prevHeartbeatResponse == null) {
    // This is the first heartbeat from the old tracker to the newly
    // started JobTracker
    if (hasRestarted()) {
      addRestartInfo = true;
      // inform the recovery manager about this tracker joining back
      recoveryManager.unMarkTracker(trackerName);
    } else {
      // Jobtracker might have restarted but no recovery is needed
      // otherwise this code should not be reached
      LOG.warn("Serious problem, cannot find record of 'previous' " +
               "heartbeat for '" + trackerName +
               "'; reinitializing the tasktracker");
               //Modified by Sayan Kole
      HeartbeatResponse myResponse1 = new HeartbeatResponse(responseId,
                    new TaskTrackerAction[] {new ReinitTrackerAction()});
                    LOG.info("Sayan setting delay myResponse1 "+ 3);
      return myResponse1;
      //end
    }
    } else {
     prevdelay=prevHeartbeatResponse.getDelay();
    if (prevHeartbeatResponse.getResponseId() != responseId) {
      LOG.info("Ignoring 'duplicate' heartbeat from '" +
          trackerName + "'; resending the previous 'lost' response");
                        //      LOG.error("sayan in 05 if delay " +
              delay + " resp del "+ response.getDelay());

      return prevHeartbeatResponse;
    }
  }
}
// Process this heartbeat
short newResponseId = (short)(responseId + 1);
int newdelay=delay;
status.setLastSeen(now);

if (!processHeartbeat(status, initialContact, now)) {
  if (prevHeartbeatResponse != null) {
    trackerToHeartbeatResponseMap.remove(trackerName);
  }

  //return new HeartbeatResponse(newResponseId,
   //           new TaskTrackerAction[] {new ReinitTrackerAction()});

//Modified by Sayan Kole
  HeartbeatResponse myResponse2 = new HeartbeatResponse(newResponseId,
          new TaskTrackerAction[] {new ReinitTrackerAction()});
  LOG.info("Sayan setting delay myResponse2 "+ 3);
  //myResponse2.setDelay(2);
  return myResponse2;
  //end
}
            //LOG.error("sayan in 1 if delay " + delay + " resp del "+
                response.getDelay());

// Initialize the response to be sent for the heartbeat
HeartbeatResponse response = new HeartbeatResponse(newResponseId,
    null);
List<TaskTrackerAction> actions = new ArrayList<TaskTrackerAction>();
boolean isBlacklisted = faultyTrackers.isBlacklisted(status.getHost());
//Modified Sayan Kole
String host = status.host;
float hval;
LOG.error("Sayanko Host is " + host);
```

```
test.setInputFile("Dmatrix.xls");
test.read();
hval=sayanwait.getHvalue(host);
LOG.error("*Sayan* In JobTracker:heartbeat, Hvalue is " + hval +
    ",thres is " + sayanwait.thres + "\n");
/*if(hval < sayanwait.thres){
    response.setDelay(0);
}
else {
    delay--;
    response.setDelay(delay);
}*/
// Check for new tasks to be executed on the tasktracker
if (recoveryManager.shouldSchedule() && acceptNewTasks &&
    !isBlacklisted) {
  TaskTrackerStatus taskTrackerStatus =
      getTaskTrackerStatus(trackerName);
            //LOG.error("sayan in 03 if delay " + delay + " resp del "+
                response.getDelay());

  if (taskTrackerStatus == null) {
    LOG.warn("Unknown task tracker polling; ignoring: " + trackerName);
  }
  else {
  List<Task> tasks = getSetupAndCleanupTasks(taskTrackerStatus);
  if (tasks == null ) {
                          if (hval > sayanwait.thres && delay>0){
                                 LOG.error("sayan in 05 if prev delay "
                                     + prevdelay);

                              LOG.error("sayan hval>thres and delay>0
                                  hval "+hval + " delay " + delay );
                            LOG.info("Hello sayanflag is: "+ sayanflag);
                              if(sayanflag==1){
                                  newdelay--;
                                  response.setDelay(newdelay);
                              }
                              else
                                  response.setDelay(newdelay);

                          }
                          else {
                              LOG.error("sayan either hval<thres or
                                  delay=0 hval "+hval + " delay " +
                                  delay );
                              //response.setDelay(0);
                             tasks =
                           taskScheduler.assignTasks(taskTrackers.get(trackerName));
                          }
                  }
```

```
                    if (tasks != null) {
                            LOG.debug("Tasks is not null right now ");
                            response.setDelay(newdelay);
                            for (Task task : tasks) {
                                    LOG.debug("I am in for loop sayan ");
                                    expireLaunchingTasks.addNewTask(task.getTaskID());
                                    if(LOG.isDebugEnabled()) {
                                            LOG.debug(trackerName + " ->
                                                LaunchTask: " +
                                                task.getTaskID());
                                    }
                                    actions.add(new
                                       LaunchTaskAction(task));
                                    newdelay--;
                                    //flag=1;
                                    response.setDelay(newdelay);
                            }
                    }
            }

    }
    // Check for tasks to be killed
    List<TaskTrackerAction> killTasksList = getTasksToKill(trackerName);
    if (killTasksList != null) {
      actions.addAll(killTasksList);
    }

    // Check for jobs to be killed/cleanedup
    List<TaskTrackerAction> killJobsList = getJobsForCleanup(trackerName);
    if (killJobsList != null) {
      actions.addAll(killJobsList);
    }

    // Check for tasks whose outputs can be saved
    List<TaskTrackerAction> commitTasksList = getTasksToSave(status);
    if (commitTasksList != null) {
      actions.addAll(commitTasksList);
    }

    // calculate next heartbeat interval and put in heartbeat response
    int nextInterval = getNextHeartbeatInterval();
    response.setHeartbeatInterval(nextInterval);
    response.setActions(
                    actions.toArray(new
                       TaskTrackerAction[actions.size()]));

    // check if the restart info is req
    if (addRestartInfo) {
      response.setRecoveredJobs(recoveryManager.getJobsToRecover());
    }
```

```
//Added by Sayan Kole
LOG.error("*Sayan* In JobTracker:heartbeat, new delay is " +
   response.getDelay() + "\n");
   LOG.error("*Sayan* In JobTracker:heartbeat, new resID is" +
      response.getResponseId());
   LOG.error("*Sayan* In JobTracker:heartbeat, new interval is" +
      response.getHeartbeatInterval());


   try {
      BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new
         FileOutputStream("FT_logger", true)));
      out.write(System.currentTimeMillis() + "," + host + "," + hval +
         "," + response.getDelay() + "\n");
      out.flush();
      out.close();
      }
      catch (Exception ex)
      {
            LOG.error("In JobTracker:heartbeat, log temperature and
               frequency, " + ex.getMessage());
      }


   //End

   // Update the trackerToHeartbeatResponseMap
   trackerToHeartbeatResponseMap.put(trackerName, response);

   // Done processing the hearbeat, now remove 'marked' tasks
   removeMarkedTasks(trackerName);

   return response;
  }
```

A.4. MODIFIED CODE FOR TASKTRACKER.JAVA

```
// TaskTracker.java

public void setDelay(){
         int delmax=700,delmed=400;
            Runtime run = Runtime.getRuntime();
            Process pr = run.exec("ipmitool sdr | grep Temp | grep \"
               CPU \" | awk -F\\| '{print $2;}' | awk '{print $1}'");
            pr.waitFor();
            BufferedReader buf = new BufferedReader(new
                InputStreamReader(pr.getInputStream()));
            String line = "";
            line=buf.readLine();
            int temp = Integer.parseInt(line);
            if(temp >38){
                    Delay=delmax;
            }
            if(temp<38 && temp>30){
                    Delay=delmed;
            }
            if(temp<30){
                    Delay=0;
            }
      LOG.info("Sayan is setting delay in setDelay " + Delay);


}
```