Cooling Hadoop: Temperature Aware Schedulers in Data Centers

by

Sanjay Kulkarni

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 4, 2013

Keywords: Hadoop, HDFS, MapReduce, Temperature, Scheduler, Distributed Computing, Cloud Computing, Cluster, Data Center

Approved by

Xiao Qin, Chair, Associate Professor of Computer Science and Software Engineering
Alvin Lim, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Associate Professor of Computer Science and Software Engineering

Abstract

The amount of unstructured data, also known as Big Data in Internet is growing every day. Because the Big data is unstructured, a large-scale distributed batch processing infrastructure like Hadoop is used instead of traditional databases. Hadoop is an open source framework, which uses MapReduce programming model to process large data set.

Hadoop's true power lies in while working in a cluster of machines in data centers. Hadoop's master-slave architecture enables master node to control the slave nodes to store and process the data. When a client application submits a job to Hadoop, the scheduler in master node schedules tasks on every available slave to process the job in parallel fashion. Many existing Hadoop schedulers do not consider the nature of the job, workload, power and temperature distribution in the data center, which is very critical and important to improve life of devices and cut down on cooling costs, which is about 25% of total investment in data centers.

Based on thorough investigations of Hadoop's existing schedulers, we propose a couple of new thermal aware schedulers that schedules tasks to balance the outlet temperature across all nodes and reduce AC costs in data center. First is a dynamic scheduler, which schedules a job based on the CPU and disk's temperature and utilization feedback given by all slave nodes at run-time. Second is a static scheduler, which assigns tasks to slaves based on CPU and disk's temperature and stored job information. Both these schedulers are implemented on top of Hadoop's FIFO scheduler. We test our schedulers and FIFO scheduler by running a set of standard Hadoop benchmark applications like WordCount, DistributedGrep, PI at different temperature, utilization thresholds and cluster sizes. The experimental results show that our schedulers achieve average outlet temperature saving of $\sim 2^{\circ}$C over the default FIFO scheduler that saves about 15% of cooling cost with little performance overhead.

Acknowledgments

There are many people in Auburn who deserve my gratitude for helping me pursue my M.S dreams. Foremost among them is Dr. Xiao Qin, who has truly been an outstanding adviser. Without his broad vision and continuous support, this thesis would never have been possible. I shall forever remain indebted to him for his guidance in my research and my career. I would also like to thank Dr. Saad Biaz and Dr. Alvin Lim for serving as members of my advisory committee.

I also owe much gratitude to Dr. Daniela Marghitu, Dr. David Umphress and Dr. Kai Chang for shaping my graduate student career for the better. I would also like to acknowledge the efforts of Ms. Michelle Wheeles, Ms. Jo Lauraitis, Ms. Carol Lovvorn and Ms. Penny Christopher in helping me keep my school and immigration paper work in order. My thanks also go out to my colleagues at Shelby 3139 and Shelby 2104. In particular, I would like to thank the group of Ajit Chavan, Yun Tian, Ji Zhang, Xunfei Jiang and Tausif for suggestions and help. I am also deeply indebted to the families of Dr. Dave Sree and Mr. Nagaraj Ejantkar for ensuring that I missed none of the festivals celebrated back home. In addition to these families, I would also like to thank my brother Santosh Kulkarni, my friends Adarsh Jain, Harish Rao, Vijay Sheshadri, Abilash Kittanna, Prateek Hejmady, Nitilaksh Hiremath, Amith Jain, Poojita Puligundla, Kanika Grover, Swathi Dumpala, Ramaraju Yelavarthy, Harsha Banavara, Pratap Simha, Deepika Rao and Rakshith Venkatesh for all their support, laughs and companionship.

Above all, I would like to express my deepest gratitude to my family for their love, compassion and support in my endeavor. Together they define my existence and it is to them that I lovingly dedicate this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing often referred to as simply the cloud, is leading emerging utility computing, which provides the basic level of computing service that is considered essential to meet the everyday needs of the general community. Gartner Inc. have predicted that at year-end 2016, more than 50% of Global 1000 companies will have stored customer-sensitive data in the public cloud [12]. IDC have predicted that 80% of new commercial enterprise apps will be deployed on cloud platforms [11]. Without any doubts we can say that term cloud computing phrase has become "du jour" of the computing world.

Although the cloud computing has existed now for over a decade, it is still a new business model in computing world and lacks proper definition. IBM in [9], defines Cloud computing as the delivery of on-demand computing resources, which includes everything from applications to data centers, over the Internet and on a pay-for-use basis. Gartner in [4], defines Cloud computing as a style of computing where scalable and elastic IT-related capabilities are provided as services to external customers using Internet technologies. NIST (National Institute of Standards and Technology) in [15], defines Cloud Computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider's interaction.

Cloud computing has been very popular with small and medium business where Cloud computing gives access to the software, technologies and other services for low cost. Cloud computing can save money when the service provider provides fast machines, installs software, maintains and runs it for client without having client to invest individually on each

of them. Clients also save on investments like the IT support, hardware space, maintenance and cost of infrastructure to support the hardware. The high availability trait of cloud computing allows clients to access the software applications from anywhere using a computer system that is connected to the internet. On the performance side, the clients get access to the powerful distributed or grid computing infrastructure which is very useful for jobs which require complex calculations and large data processing.

According to the Software Engineering Institute[22], the cloud computing environments can either be public or private. These computing environments represent the way the services are offered to the clients. In public environment the services are offered to clients either free or for a fee. The private environments are generally limited to organizations in which services are deployed behind the organization's firewall. The computing environments should take one of the 3 popular service models as described below:

- Infrastructure-as-a-Service (IaaS): The computational infrastructure includes of a set of virtual machines that have computation and storage capacity and are available for access over the Internet. In this model, clients can run computation intensive or a data intensive job using a variety of interfaces that facilitate the interaction. The services are provided over an infrastructure and client's programs do not have rights to access or modify it. Some examples of IaaS include: Amazon Cloud Formation, Rackspace Cloud and Google Compute Engine.

- Platform-as-a-Service (PaaS): This service model provides the basic platform upon which the clients can write their own applications and deploy it. The platform includes operating systems, libraries, environments, services, supporting tools provided by the service provider. Like IaaS, PaaS also does not allow user programs to alter the underlying infrastructure. However, they can modify and change the settings that are in application's scope and environment [14].

- Software-as-a-Service (SaaS): In this service model, Software developed by the client, provider or by a third party is provided as service to the client. The client do not run the application locally, instead it would use an API to communicate with the application that runs in the cloud platform remotely. These APIs cannot modify the underlying cloud infrastructure, however they can still be used to customize and change the application's configuration. A few examples of SaaS include GMail, GDocs and Office 365.

Along with service models like Iaas and PaaS, MapReduce and BigData processing are growing in popularity every day. In fact, popular IaaS models like Google Compute Engine are used along with the distributed processing framework like Apache Hadoop which implements MapReduce framework to process and analyze the Big Data. Microsoft Azure, Amazon EC2 also use different distributions of Hadoop to implement PaaS and deploy applications. There are several real world example and prototypes of PaaS and IaaS being built entirely on the Hadoop framework [23, 20]. It is believed that Cloud Computing based on Hadoop will be the next big trend in the IT industry.

To supply the demand for large computing power and storage space in service models like IaaS, PaaS and SaaS, the service providers are inclined towards creating new data centers. Data centers are facilities that host hundreds of thousands of servers which concurrently support a myriad of distinct services and applications[14]. With the growth of private cloud, the enterprises already having the traditional data center infrastructure want to use their data centers with Cloud. That helps the service providers save cost by bulk deployment, instill cloud infrastructure and help them to easily manage and maintain the resources. Also, flexibility of cloud helps them to easily consolidate discrete cloud environments with discrete data centers and morph data centers to much more efficient and effective infrastructure than its current state of affairs[16].

The amount of unstructured data, also known as Big Data in Internet is growing every day. In 2012 Big data is expected to grow 48% larger to 2.7 Zetta (2.7*1021) bytes annually

and by 2015 it is expected to triple to 8 Zeta bytes given the rising popularity of sites like Facebook, Twitter, Amazon and YouTube. The Big data is large and unstructured, so it is really hard to process and analyze using the traditional database models like RDBMS. In cloud computing, MapReduce is the programming model used for processing and analyzing large data sets. Google introduced MapReduce[1]: a large-scale distributed batch processing infrastructure to analyze and process these large data sets. MapReduce programming model is used in Apache's Hadoop as well which is open sourced software framework for processing large data. Hadoop could actually process "web-scale" data on the order of hundreds of gigabytes to terabytes or petabytes[30]. Hadoop is designed to efficiently process large volumes of information by connecting many commodity computers together to work in parallel. Hadoop uses the Hadoop Distributed File System to store the large data and provide streaming access to the data to every node in the cluster while working in parallel. MapReduce and HDFS together give Hadoop the power to process and analyze the data. Hadoop is being used in areas of web crawling, analytics machine learning, image processing and data mining which have huge jobs that are typically handled in data centers having tens of thousands of computers.

1.1 Overview of MapReduce Framework in Hadoop

MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. The model does not work by sharing the data arbitrarily between the nodes. Instead, the data elements in the MapReduce are immutable. The data is written only once and read many times. The data read from the input files in HDFS are processed and converted to intermediate values and further processed to generate outputs. Any changes on the input files during this process are not reflected on the actual files.

[1]"MapReduce" is the name for both programming model and distributed processing infrastructure introduced by Google. To disambiguate, we further refer distributed programming infrastructure with name "MapR" and as is for programming model.

As the name suggests MapReduce programs process the input data in two stages- Map stage and Reduce stage. In the mapping stage, the mapper takes one item at a time from the input list of data elements that are fetched from the HDFS and transforms to an intermediate output data element. The Map operations are paralleled when input file set is first split to several pieces called File Splits or Input Splits. Every mapper would have exactly one input split; the number of mappers created is dependent on the number of input splits. Splitting the input file set helps in paralleling the processing as the mappers do not have to synchronize and contend to read the file. Moreover, mappers do not have any identities of their own, so all mappers are the same and are not aware of each other's existence let alone communication taking place between them.

Every mapper that receives the input split processes it in a specified format. The input split parser (or Record Reader) in the mapper parses the split and generates the key-value pairs. The key-value pairs are processed in parallel by the mappers, one at a time to generate exactly one intermediate key-value pair for every (key,value) pair. The output (key,value) pair of the mapper serves as input to the reducer.

When the mapping phase has completed, the intermediate (key, value) pairs must be exchanged between machines to send all values with the same key to a single reducer. The reducer receives the intermediate data generated by the mapper as input, combines the values of all mapper outputs and generates a single output data corresponding to the input data fetched by the mapper. The reducers reduce a key value that is unique to each other, so reducers are same as mappers in the sense that they do not have to communicate with each other and also remain anonymous to each other.

1.2 Overview of Hadoop Distributed File System

The Network File System (NFS) is the most commonly used distributed file system, where the file system exists on some node in a network. The filesystem appears as the single drive and its storage capacity is limited to the HDD capacity of that node, which means

that large amount of information cannot be stored in NFS. However, to store the peta or tera bytes of data we need to add a node with HDD to the network, but the filesystems in both these nodes would be discrete and would appear as discrete drives in the client. For example, to store a file of 1TB size in NFS having 2 nodes of 500GB storage each, we have to split the 1TB file to 2 different files and store each file in the 2 nodes in network. So, when the client requests a read/write to the file, it has to manage 2 different file pointers or more if the file were any bigger and the NFS had more storage nodes. On the other hand, if there are many clients trying to access information in one file, the server gets overloaded with requests and filesystem throughput drops significantly. Second drawback of using the NFS is that it is not reliable, i.e if the storage node goes down then the information is unavailable. Last important drawback of NFS is that the clients should always copy the data to the local machines before they can operate on it.

Hadoop Distributed File System (HDFS) is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware[7]. It is designed to run in user space making it extremely portable across platforms. For the sake of dealing with large files, it provides high throughput access of data to application rather than low latency access. Like any other local file system, HDFS also divides the files in the file system to blocks, but block sizes in HDFS are larger in contrast to local file system blocks. Finally to improve reliability, HDFS replicates the data by a factor as configured by user and it uses strategies to replicate the data in an efficient manner to improve availability of data and reduce network bandwidth utilization.

Like MapReduce, HDFS implementation also follows the master/slave architecture in the cluster. The master node known as NameNode manages the system namespace and regulates the access to files by clients. In addition there are DataNodes, which are the slaves, and usually exist in every node in the cluster. The NameNode is responsible to open, close, rename the files and DataNodes are responsible for storing, reading, writing data and follow the instructions of NameNode [7]. In addition to the NameNode and the DataNode

there is a Secondary NameNode to improve reliability and fault tolerance which acts as NameNode when the primary NameNode fails.

1.3 Reliability and Fault tolerance in MapReduce and HDFS of Hadoop

The user never explicitly marshals information from one machine to another; all data transfer is handled by the Hadoop MapReduce platform itself, guided implicitly by the different keys associated with values [32]. This is fundamental element of Hadoop MapReduce's reliability. Hadoop gets periodic reports from the DataNodes or TaskTrackers which helps the master node to identify a failed node and restart the task assigned to it. If the failed nodes have been performing side-effects tasks, e.g., communicating with the outside world, then the shared state must be restored in a restarted task. By eliminating communication and side-effects, restarts can be handled more gracefully.

1.4 Job Schedulers in Hadoop

The performance of a master-worker system like MapReduce system closely ties to its task scheduler on the master. Hadoop schedulers are designed as jar module and can be easily plugged in to any Hadoop distro. Although there have been lot of work on schedulers, they are still in the early stages of its life compared to OS's schedulers. Still, there are quite a few popular schedulers that are worth mentioning:

- The FIFO scheduler is the default scheduler in Hadoop which uses a single queue for scheduling tasks (partitioned jobs) with a FIFO method.
- Yahoos capacity scheduler uses multiple queues for scheduling. It schedules jobs and assigns resources to jobs based on resources capacity allocated for the queue of jobs and usage density density of capacities.
- Facebooks fair scheduler uses multiple queues for allocating different resources in the cluster. The fair scheduler maintains a pool of jobs with each pool having a dedicated

number of Map and Reduce slots. It runs a job by using the map and reduce slots and if a pool is not running any job then the free slots can be allocated to other pools.

- Dynamic Priority Scheduler is a parallel task scheduler in which it allows users to control their allocated capacity by adjusting their spending over time.

1.5 Need for thermal management

Although data centers running Map-Reduce framework are effcient in processing Big-Data, it comes at the cost of investments in various forms like realty, electricity, servers, maintenance etc. In recent years big companies like Facebook, Microsoft and Google have invested billions of dollars in just maintaining the infrastructure of the data center supporting cloud services. The high maintenance cost is predominantly due to high electricity and cooling costs, which is 25% of the total investment. In fact, the cooling costs of a data center are higher than the entire IT equipment it supports [3].

Tremendous amount of data storage, computation power and the access to the data centers nowadays result in high power consumption on servers in data centers. The demand for data and short job latency result in high CPU and disk utilization. High utilization over a period of time gets CPU and disk heated up and cooling systems work hard to normalize the heat generated and hence increases the cooling costs. So, it becomes extremely important to manage temperature and power in a data center. There are a number of works on thermal and power management of data centers. Some works are based on developing a scheduler for temperature balancing, workload balancing and computing energy minimization. Some works reduce the heat recirculation and some techniques develop a scheduler to manage in constant and linear cooling model.

Most of the schedulers in Hadoop have concentrated on looking at the scheduling problem from the masters perspective, where the scheduler on the master node tries to assign equal work across all the worker nodes[33]. None of the Hadoop schedulers developed so far considers the nature of the job, the power consumption of the node and most importantly

the thermal model of the data center while scheduling the jobs. Although there have been many works on scheduling the tasks in the data center to make data center more thermal aware, none of the schedulers have been implemented in Hadoop to see their performance in reality.

1.6 Motivation

Many thermal aware algorithms are primarily simulation based. There are sufficient evidences to prove that Computational Fluid Dynamics is too complex and not suitable for Online scheduling [26] [18]. Most of the algorithms do not consider the nature of the tasks, a task might be a CPU intensive task, or it might be a data intensive task. The CPU and the disk temperature rise by scheduling the job on a node is ignored. The algorithms do not have means to manage the temperature and the performance if the temperature of the nodes if they are above certain threshold. Finally, the algorithms are not implemented in Hadoop and may not be accurate for Hadoop.

1.7 Our Contribution

Based on thorough investigations of the shortcoming of Hadoop's existing schedulers, we propose a couple of new thermal aware schedulers that schedules tasks on the slave nodes with only intention of balancing the temperature across all nodes and reduce AC costs in data center. We formulate an algorithm for a dynamic scheduler, which schedules the job based on the CPU and disk temperatures and utilization feedback given by the slave nodes. Second is a static scheduler, which assigns a job to slave based on its CPU and disk temperature and stored profile information of job of same kind. Both these schedulers are implemented on top of Hadoops FIFO scheduler. We run a set of standard Hadoop benchmarks and application like word count, distributed grep, pi at different temperature thresholds, cluster sizes and utilization on our schedulers and Hadoop's FIFO scheduler. The experimental results show

that our schedulers achieve average outlet temperature saving of 1-2°C over the default scheduler which saves about 15%-10%of cooling cost with little performance overhead.

1.8 Organization

This thesis is organized as follows. Chapter 2 explains in detail the Hadoop's architecture, HDFS and MapReduce framework in Hadoop. Chapter 3 explains the problem and explains existing solutions. Chapter 4 describes design of our thermal aware scheduler. Chapter 5 analyzes the results and performance of our schedulers with the Hadoop's default FIFO scheduler. Chapter 6 refers to future work and concludes the thesis.

Chapter 2

Hadoop

Apache Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Rather than relying on high-end hardware, the resiliency of these clusters comes from the softwares ability to detect and handle failures at the application layer. Hadoop enables a computing solution that is scalable, cost effective, flexible and fault tolerant [10].

2.1 Hadoop Architecture

Hadoop is implemented using relatively simple model of Client-Master-Slave design pattern. There are two masters in the architecture, which are responsible for the controlling the slaves across the cluster. The first master is the NameNode, which is dedicated to manage the HDFS and control the slaves that store the data. Second master is JobTracker, which manages parallel processing of HDFS data in slave nodes using the MapReduce programming model. The rest of the cluster is made up of slave nodes which runs both DataNode and TaskTracker daemons. DataNodes obey the commands from its master NameNode and store parts of HDFS data decoupled from the meta-data in the NameNode. TaskTrackers on the other hand obeys the commands from the JobTracker and does all the computing work assigned by the JobTracker. Finally, Client machines are neither Master or a Slave. The role of the Client machine is to give jobs to the masters to load data into HDFS, submit Map Reduce jobs describing how that data should be processed, and then retrieve or view the results of the job when its finished.

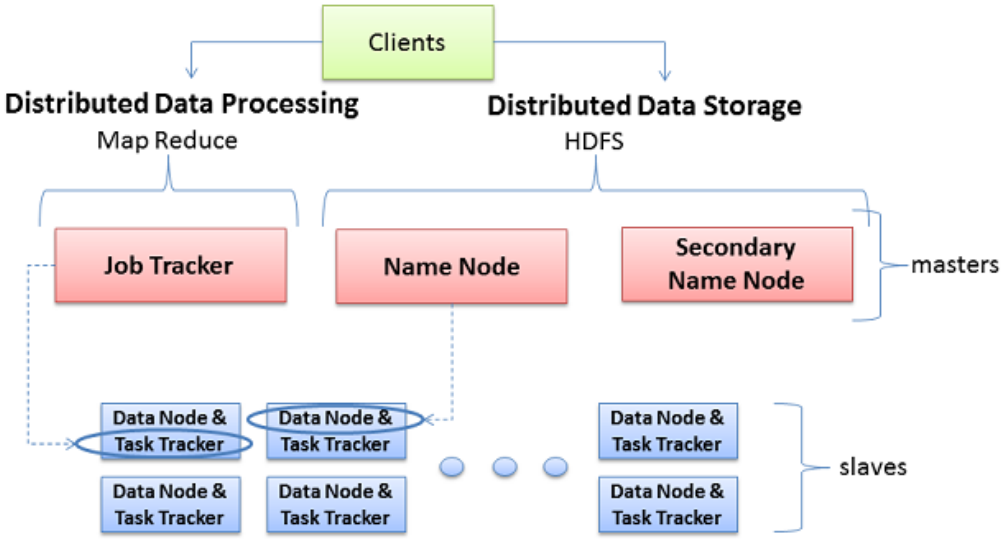

Figure 2.1: Hadoop Architecture

Figure 2.1 [8] shows the basic organization of the Hadoop cluster. The client machines communicates with the NameNode to add, move, manipulate, or delete files in HDFS. The NameNode in turn calls the DataNodes to store, delete or make replicas of data being added to HDFS. When the client machines want to process the data in the HDFS, they communicate to the JobTracker to submit a job that uses MapReduce. JobTracker divides the jobs to map/reduce tasks and assigns it to the TaskTracker to process it.

Typically, all nodes in Hadoop cluster are arranged in the air cooled racks in a data center. The racks are linked with each other with the help of rack switches which runs on TCP/IP.

2.2 HDFS

Hadoop Distributed File System is the filesystem designed for Hadoop to store the large sets of data reliably and stream those data to the user application at the high throughput rather than providing low latency access. Hadoop is designed in Java and that makes it incredibly portable across platform and operating systems. Like the other distributed file systems like Lustre and PVFS, HDFS too stores the meta data and the data separately. The

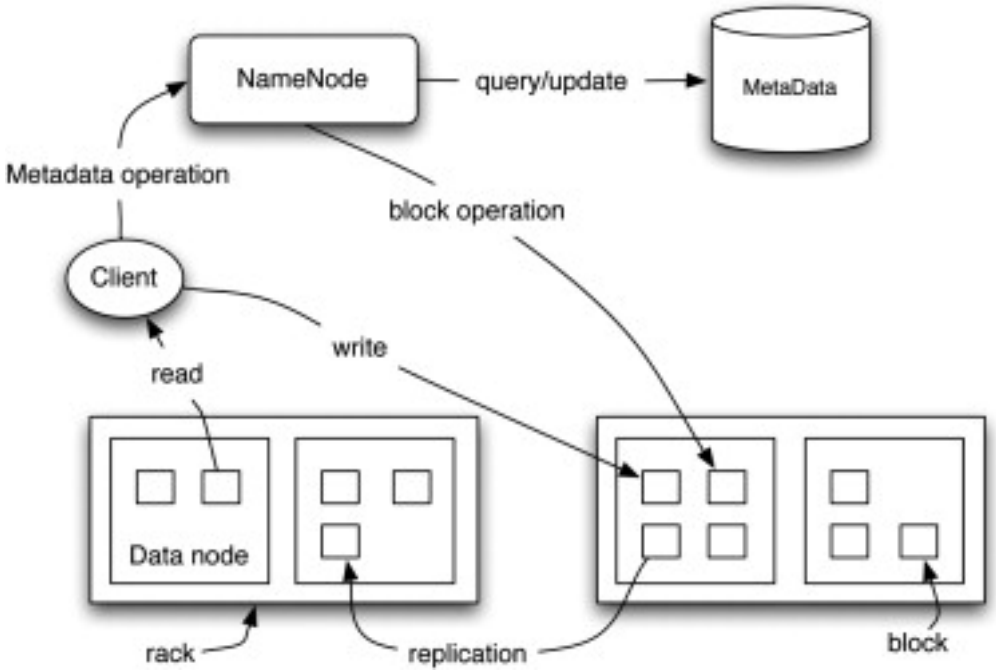

Figure 2.2: HDFS Architecture

NameNode stores the meta-data and the DataNodes store the application data. But, unlike Lustre and PVFS, the HDFS stores the replicas of the data to provide high throughput data access from multiple sources and also data redundancy increases the fault tolerance of HDFS.

When the HDFS replicates it does not replicate the entire file, it divides the files into fixed sized blocks and the blocks are placed and replicated in the DataNodes. The default block size in Hadoop is 64MB and is configurable.

2.2.1 NameNode, DataNode and Clients

The Figure 2.2 [28] shows the HDFS architecture in Hadoop which contains three important entities- NameNode, DataNode and Client. The NameNode is responsible for storing the meta-data, and track the memory available and used in all the DataNodes. The client which wants to read the data in the HDFS first contacts the NameNode. The Namenode then looks for the block's DataNode which is nearest to the client and tells the client to access the data from it. Similarly, when the client wants to write a file to the HDFS, it

requests the NameNode to nominate 3 DataNodes to store the replicas and the client writes to it in streamline fashion. The HDFS would work efficiently if it stored the files of larger size, at least size of a block because the HDFS stores the Namespace RAM. If it were all smaller files in HDFS then the inodes information would occupy the entire RAM leaving no room for other operations.

The NameNode would register all the DataNodes at the start-up based on the NamspaceID. The NameSpaceID would be generated when the NameNode formats the HDFS. The DataNodes are not allowed to store any blocks of data if the NamespaceID does not match with the ID of the NameNode. Apart from the registering the DataNodes in the start-up the DataNodes send the block reports to the NameNode periodically. The block report contains the block id, the generation report and the length of the each block that DataNode holds. Every tenth report sent from the DataNode is a block report to keep the NameNode updated about all the blocks. A DataNode also sends the HeartBeat messages that just notify the NameNode that it is still healthy and all the blocks in it are intact. When the NameNode does not receive a heartbeat message from the DataNode for about 10 seconds, it assumes that the DataNode is dead and uses it's policies to replicate the data blocks in the dead node to other nodes that are alive.

Similar to most conventional file systems, HDFS supports operations to read, write and delete files, and operations to create and delete directories. The user references files and directories by paths in the namespace. The user application generally does not need to know that file system metadata and storage are on different servers, or that blocks have multiple replicas.

2.2.2 Backup Node and Secondary NameNode

The NameNode is the single point of failure for the Hadoop cluster, so the HDFS copies the of the Namespace in NameNode periodically to a persistent storage for reliability and this process is called checkpointing. Along with the NameSpace it also maintains a log of

the actions that change the Namespcace, this log is called journal. The checkpoint node copies the NameSpace and journal from NameNode to applies the transactions in journal on the Namespace to create most up to date information of the namespace in NameNode. The backup node however copies the Namespace and accepts journal stream of Namespace and applies transactions on the namespace stored in its storage directory. It also stores the upto-date information of the Namespace in memory and synchronizes itself with the NameSpace. When the NameNode fails, the HDFS picks up the Namespace from either BackupNode or CheckPointNode.

2.2.3 Replica and Block Management

HDFS makes replicas of a block with a strategy to enhance both the performance and reliability. By default the replica count is 3, and it places the first block in the node of the writer, the second is placed in the same rack but different node and the third replica is placed in different rack. In the end, no DataNode contains more than one replica of a block and no rack contains more than two replicas of same block. The nodes chosen on the basis of proximity to the writer, to place the blocks.

There are situations when the blocks might be over-replicated or under-replicated. In case of over-replication the NameNode deletes the replicas within the same rack first and from the DataNode, which has least available space. In case of under-replication, the NameNode maintains a priority queue for the blocks to replicate and the priority is high for the least replicated blocks.

There are tools in HDFS to maintain the balance and integrity of the data. *Balancer* is a tool that balances the data placement based on the node disk utilization in the cluster. The *Block Scanner* is a tool used to check integrity using checksums. *Distcp* is a tool that is used for inter/intra cluster copying.

2.3 MapReduce

In Introduction chapter we understood that MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. MapReduce programs are influenced by functional programming constructs used for processing lists of data. The MapReduce fetches the data from the HDFS for parallel processing. These data are divided in to blocks as mentioned in the section above.

2.3.1 JobTracker and TaskTracker

JobTracker is the master, to which the applications submit MapReduce jobs. The JobTracker gets the map tasks based on input splits and assigns tasks to TaskTracker nodes in the cluster. The JobTracker is aware of the data block location in the cluster and machines which are near the data. The JobTracker assigns the job to TaskTracker that has the data with it and if it cannot, then it schedules it to the nearest node to the data to optimize the network bandwidth. The TaskTracker sends a HeartBeat message to the JobTracker periodically, to let JobTracker know that it is healthy, and in the message it includes the memory available, CPU frequency and etc. If the TaskTracker fails to send a HeartBeat to the JobTracker, the JobTracker assumes that the TaskTracker is down and schedules the task to the other node which is in the same rack as the failed node.

The Figure 2.3 [31] shows the data flow of MapReduce in couple of nodes . The steps below explains the flow of the MapReduce [31].

1. Split the file: First the data in the HDFS are split up and read in *InputFromat* specified. InputFormat can be specified by the user and any InputFormat chosen would read the files in the directory, select the files to be split into InputSplits and give it to RecordReader to read the records in (key, value) pair that would be processed in further steps. Standard InputFormats provided by the MapReduce are

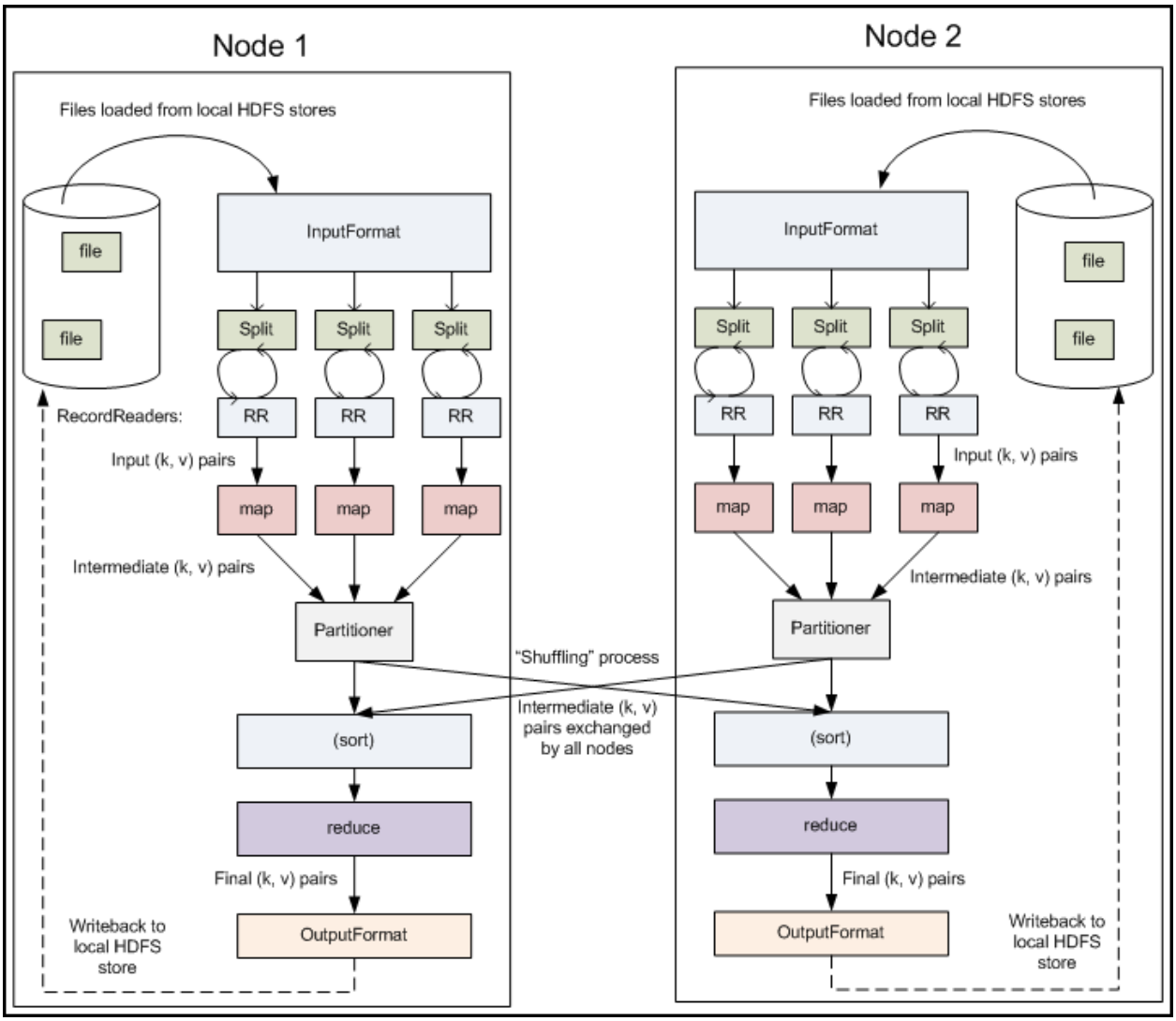

Figure 2.3: MapReduce data flow.

- *TextInputFormat* reads text files where the byte offset is key and line contents is value.
- *KeyValueInputFromat* reads (key,val) pair. Keys and values are separated with a ¡*tab*¿ key.
- *SequenceFileInputFormat* is Hadoop specific high-performance binary format where key and value are user defined.

The InputSplit is the unit work that comprises a single map task in a MapReduce program. The job submitted by the client is divided into the number of tasks, which is equal to the number of InputSplits. The default InputSplit size is 64MB and can be configured by modifying split size parameter. The InputSplits enable the parallel processing of MapReduce by scheduling the map tasks on other nodes in cluster at same time. When the HDFS splits the file into blocks, the task assigned to that node accesses the data locally.

2. Read the records in InputSplit: The InputSplit although is ready to be processed it still does not make sense to the MapReduce program as the input to it is not in key-value format. The *RecordReader* actually loads the data and converts it to ¡key,value¿ pair expected by the Mapper task. The calls to RecordReader calls map() method of Mapper

3. Process the records: When the Mapper gets the key-value pair from the RecordReader, it calls the map() function to process the input key-value pair and output an intermediate key-value pair. While these mappers are reading their share of data and processing it in parallel fashion across the cluster, they do not communicate with each other as they have no data to share. Along with the key-value pair, the Mapper also gets couple of objects, which indicates where to forward the output and report the status of task.

4. Combiner[1] combines all the ¡key,value¿ pair with same keys before sending intermediate data to the Reducer. It is in some ways a mini Reducer.

5. Partition and Shuffle: The mappers output the key,value pair which is the input for the reducer. This stage the mappers begin exchanging the intermediate outputs and the process is called shuffling. The reducer reduces the intermediate value with the same key and it partitions all the intermediate output with the same key. The partitioner determines which partition a given ¡key,value¿ pair go to. The intermediate data are sorted before they are presented to the Reducer.

6. Reduce the mapper's output: For every key in the assigned partition in the reducer a reduce() function is called. Because the reducer reduces the partition with the same key, it iterates over the partition to generate the output. The OutputFormat will specify the format of the output records, and the reporter object reports the status. The RecordWriter writes the data to file specified by the OutputFormat.

[1]This is an optional step and absolutely used for optimization. It is easy to implement using the Reducer interface.

Chapter 3

Motivation and Existing Solutions

In chapter 1 we saw that the data center investments include realty, electricity, servers, maintenance etc and many big companies like Facebook, Microsoft and Google have invested billions of dollars in just maintaining the infrastructure of the data center supporting cloud services. Microsoft in [5] have analyzed associated cost in a data center shown as in Table 3.1. The Servers and Network are one time investment even after considering the hardware failures. The Power and the Infrastructure is about 40% of total cost which is spent every month or every year.

Amortized Cost	**Component**	**Sub-Components**
45%	Servers	CPU, memory, storage systems
25%	Infrastructure	Power distribution and cooling
15%	Power Draw	Electrical utility costs
15%	Network	Links, transit, equipment

Table 3.1: Costs in Data Center

Although the Infrastructure is just 25% of the total cost at data center level, the cost might scale up to millions-billions of dollars because data centers have tens of thousands of nodes and each drawing tens of Mega-Watts of power at peak. Even at cost of few cents or dollars per kilowatts of power used it scales up to thousands-millions of dollars. Power Usage Efficiency = (Total facility power)/Total IT Power is a metric introduced by the Green Grid [6] estimates that for a reasonable PUE value of 1.7, the total cost of power would sum up to 9.3 million $ a year, out of which cooling cost is about 33% of total cost. So, it is extremely important to improve the thermal performance of data centers to optimize computing resources, improve reliability, improve utilization, and maximize computation capability.

3.1 Thermal model of a data center

In the data center the server nodes are arranged in the racks. The racks are installed in the raised floor which has perforated floor tiles. The Air Conditioner, Heating, Ventilation Air Conditioner (HVAC) or Computer Room Air Conditioner (CRAC) deliver the cold air from the raised floor. The cool air enters the racks from the front side and leaves from the rear end of rack. While the cool air exists the rack it picks up the heat generated by all the servers in the nodes. The heated air forms the hot zone behind the racks which is extracted back to the air conditioner intakes which is positioned above the hot zone and this heat dissipated from the hot-zone controls the CRAC supply air. The setup of a typical data center is shown in Figure 3.1, [1] below.

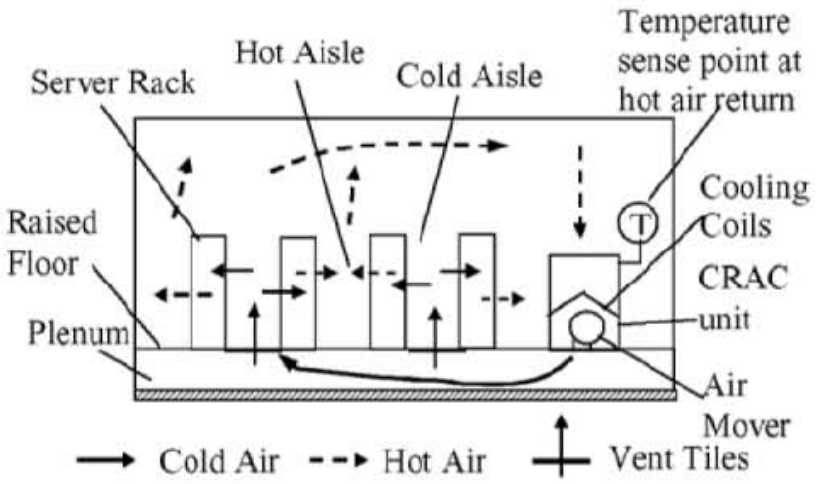

Figure 3.1: Data center layout

Many researches have begun on the dynamic optimization of the data center thermal environment. The CFD model provides the dynamics of a data center which helps considering different parameters like the co-efficient of performance, supply temperature, inlet temperature, outlet temperature, air density, computing power of a node etc. Table 3.2 will give you and overview of frequently used terms in the thermal management formulation of a data center and in this thesis.

Notion	Meaning
T_{sup}	Air temperature as supplied from cooling unit.
T_{in}	Inlet air temperature of node.
T_{out}	Outlet air temperature of a node.
T_{red}	Redline temperature of data center.
N	number of nodes in cluster.
T	Air temperature.
Q	Amount of heat carried by the air fow in unit time.
ρ	Air density. (Typical value: *1.19kg/m*^{3})
f	Flow rate. (Typical value: *520CFM = 0.2454* m^3/s)
c_p	Specific heat of air. (Typical value: *1005J/kg/K*)
P_{Total}	Total power consumed.
P_C	Sum of total computing power.
P_{AC}	Power used for cooling data center.
G	Hardware dependent specifications.
C_{Tot}	Total task set.
C_i	Total task set assigned to a node i.
C_{ut}	Total tasks assigned when temperature is below threshold.
C_{uop}	Total tasks assigned when temperature is above threshold, but below T_{CPUAvg} or $T_{DiskAvg}$.
T_{CPU}	CPU temperature.
T_{Disk}	Disk temperature.
T_{rCPU}	Redline temperature for CPU.
T_{rDisk}	Redline temperature for Disk.
T_{CPUAvg}	Average CPU temperature in cluster.
$T_{DiskAvg}$	Average Disk temperature in cluster.
U_{CPU}	CPU utilization.
U_{Disk}	Disk utilization.
U_{tCPU}	Threshold for CPU utilization.
U_{tDisk}	Trheshold for Disk utilization.
U_{CPUAvg}	Average CPU utilization in cluster.
$U_{DiskAvg}$	Average Disk utilization in cluster.
t_n	Total time take to run job.
a	power consumption of node i's power unit.
b	Power consumption of node i running a task.

Table 3.2: Table of Symbols to thermally model a data center.

In the data center layout figure 3.1 in [1], the cold air that comes from the perforated floor to cool the nodes is the supply temperature T_{sup} in the table 3.2. The supply temperature T_{sup} when it enters the room mixes with the hot air produced by the heat-zones and increases to operating temperature to the T_{in} which is called the inlet temperature. The inlet temperature is the temperature, which goes inside the node to take away the heat generated by the nodes in the data center. In data centers, the T_{sup} is set to a temperature to maintain T_{in} below a certain threshold temperature called redline temperature denoted by T_{red}. The redline temperature is the threshold temperature, above which hardware risks failure. The hot air coming out of the node is outlet temperature, represented as T_{out}. This outlet temperature is absorbed by the vents above and based on this temperature the supply temperature is further increased or decreased.

Q is the amount of heat carried by the air flow in unit time. According to the *law of energy conservation*, it is given by the equation:

$$Q = \rho f c_p T \tag{3.1}$$

where ρ is the is the air density, f is the air flow rate, c_p is the specific heat of air, and T is the air temperature.The nodes are placed in the different places in a huge data center and the air flow rate may vary for each node, so we denote it using f_i for each node, where i is the node. Similarly, the inlet and outlet temperature changes for the node, which can be given as T^i_{in} and T^i_{out}.

The total power consumed (P_{Total}) is sum of total computing power(P_C) and power used for cooling (P_{AC}). The lighting costs and other energy costs have negligible contribution to the total costs.

$$P_{Total} = P_{AC} + P_C \tag{3.2}$$

The difference between the T^i_{out} and the T^i_{in} is the heat emitted which is generated by the computation work by the node. The total power consumed for computation in the

entire cluster is the sum of the power consumed by all nodes in the cluster. It is given by the equation

$$P_c = \sum_{i=1}^{N} P_i. \tag{3.3}$$

The power consumed by the individual node in the cluster is given by

$$P_i = G_i C_i. \tag{3.4}$$

where G_i depends on the hardware specifications of the node and C_i is the task-set of the total tasks assigned to node i.

The relationship between the power consumed and the inlet and outlet temperature is given by

$$P_i = G_i(T^i_{out} - T^i_{in}). \tag{3.5}$$

substituting $\rho f_i c_p$ for G_i, we have P_i as

$$P_i = \rho f_i c_p (T^i_{out} - T^i_{in}). \tag{3.6}$$

The energy cost of air conditioning depends on the heat removed or reduced and the Coefficient Of Performance (COP) of the air conditioner. The COP in [17] is defined as the amount of heat removed by the air conditioner to the total energy consumed to remove that heat, i.e

$$COP = \frac{heat\,removed}{energy\,consumed\,to\,remove\,the\,heat} \tag{3.7}$$

The COP obtained for the water chilled CRAC unit in the HP utility data center will be considered as reference in this thesis. The COP model is given by

$$COP = (0.0068T^2_{sup} + 0.0008T_{sup} + 0.458) \tag{3.8}$$

where, T_{sup} is the supply temperature. The COP graph at different T_{sup} are given in the Figure 3.2 [17]

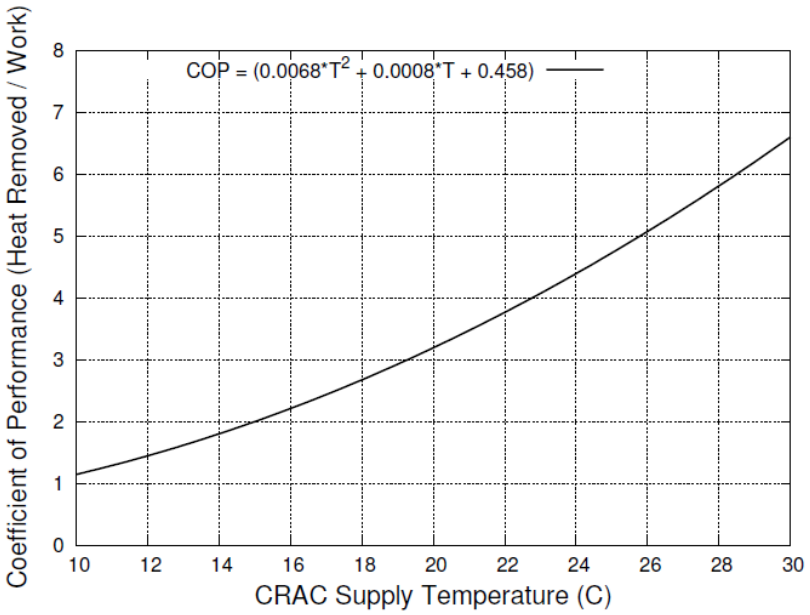

Figure 3.2: Coefficient of Performance

CoP is not linear and normally increases with the supplied air temperature. Higher the T_{sup}, higher is the COP. Higher COP means that the air conditioner does not have to work hard to cool the nodes in data center and hence, save cooling costs. The power consumed by air condition for cooling in equation can be represented in terms of computing power and COP as

$$P_{AC} = \frac{P_C}{COP} \tag{3.9}$$

substituting for P_{AC} in we have,

$$P_{Total} = (1 + \frac{1}{COP})(\sum_{i=1}^{N} G_i C_i) \; s.t \; C_{tot} = \sum_{i=1}^{N} C_i \tag{3.10}$$

In the equation the C_{tot} is the total number of tasks that has been given to Hadoop to distribute to the TaskTrackers in the node. These tasks basically boil down to few machine

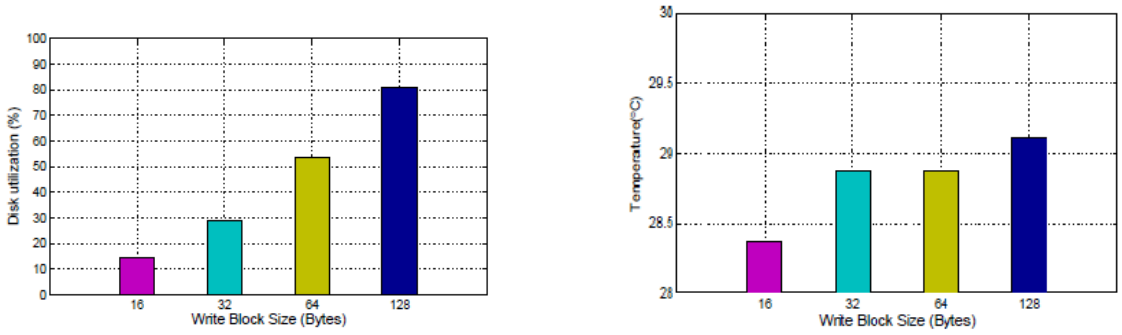

Figure 3.3: The rise in the Disk utilization increases the Outlet temperature

instructions, which take different times to execute, but at the cluster level, all TaskTrackers are executing the same instructions as they all are executing the same task. So it is safe to assume that all the TaskTrackers are utilized approximately the same and have same power consumption. Furthermore in [24], they show that there is a linear relationship between the power consumption and CPU utilization of a machine. The power consumption and the temperature rise have linear model. So from transitive property, the CPU utilization and the temperature rise have linear relationship. The CPU temperature rise has linear model with the outlet temperature.

$$\begin{aligned} P &= aC_{util} + b \\ P \ \alpha \ C_{util} \\ T_{CPU} \ \alpha \ P \\ T_{Out} \ \alpha \ T_{CPU} \end{aligned} \tag{3.11}$$

Its not just CPU temperature which contributes to the outlet temperature, in [13] they show that the increase in the Disk utilization also increases the outlet temperature by almost 1.5 degrees, as shown in the Figure 5.1. It can be observed from the Figure 5.1, the Disk utilization and temperature are not linear like CPU utilization and temperature. The relationship can be expressed as linear regression function.

The goal is to maximize T_{sup} which is achieved by minimizing the T_{out}, which would minimize the T_{in} and hence maximize the T_{sup}.

3.2 Related Work

There have been many works on thermal and power management of a data center. We will see few works that consider the thermal model of the data center.

[21] is based on Hadoop based storage data centers in which they ensure that each node in the data center operates at a temperature below the threshold (T_{Max}) and try to minimize the power consumed by AC by scheduling the tasks based on 4 constraints.

- Compatible with thermal model of data center.
- The outlet temperature is less than the threshold temperature. (T_{Max} ¡ T_{Out}
- A task is assigned only to 1 node at a time among all other nodes having the data block.
- 0 is power consumption when a node is idle, else it is p.

The algorithm basically assigns a cost for running a task on every node and calculates the T_{out} resulting from it. If the chosen system has high T_{out} then it is given to the nodes having less T_{out}, and then they calculate the T_{sup}, which satisfies the above conditions and if the new T_{sup} is higher than the current T_{sup}, task is assigned else some other node is chosen and all the steps are re-executed.

There are three thermal aware schedulers proposed in [27]. The temperature measurement is at chassis level and the task assignment granularity is at the processor level. The first algorithm is Uniform Outlet profile in which most tasks are given to the processor with low inlet temperature and fewer tasks are given to the node with high inlet temperature. The goal of the the algorithm is to balance the outlet temperature between all the nodes and reduce the recirculation of the hot air. Second algorithm is Minimal computing energy,

which tries to minimize the number of active processors. It tries to distribute the jobs to the coolest processors among the active ones. The idle processors are turned off to save energy and hence does not need to be cooled. The third scheduler distributes the job evenly among all the processors and the work done and heat generated by them is same.

HP and Duke University published works [19] [2] developed online measurement and control techniques to improve energy-efficiency of data centers. Notions like Supply Heat Index (SHI) and Return Heat Index (RHI) characterizes the energy efficiency of datacenter cooling system. They discussed that different datacenter layouts and configurations that lead to hotspots and non uniform thermal distribution. Using CFD they could find out hotspots in a data center and took remidial actions like dynamic relocation of workload to achieve thermal balancing in inside a datacenter when the datacenter is overheated.

[18] [25] algorithms reduce the Heat recirculation by distributing the task and power accordingly. The MinHR algorithms are based on calculating the HeatRecirculationFactor(HRF) for each rack. The HRF is the ratio of the HRF contributed from the rack to the sum of all other racks. If the HeatRecirculationFactor is high for a rack, then it is assigned fewer tasks compared to other racks. The HRF is assigned after several profiling runs. A small HRF indicates that rack is very huge contributor to the recirculation and assigned fewer tasks. The difference between [17] and [25] is that, the former is designed to place the power budget and latter is designed to place tasks.

Thermal Aware Task Scheduler(TASA) proposed in the [29] does not utilize the CFD model, because the CFD schedulers are hard to apply in online schedulers. Instead it uses thermal map of the data centers to schedule the jobs. The TASA scheduler schedules the jobs periodically and gets the temperature of all the nodes. It then sorts the nodes based on their temperatures from coolest-hottest. The hottest nodes which are above a pre-defined threshold are allowed to cool down for a period of time without running any tasks and later when they are cool, they are all used to schedule the hottest job.

Chapter 4

Design of Cool Schedulers

Scheduler in Hadoop primarily deigned to share the cluster between different jobs and users for better utilization of the cluster resources. Without the scheduler it is possible that a job might occupy all the resources and the other jobs might just be left waiting until this job finishes or terminates. Hadoop schedulers are designed as pluggable interface and it is possible to switch between schedulers for different jobs just by changing the scheduler's class in the configuration file. As we saw in Chapter 1, Hadoop is shipped with 3 schedulers- FIFO, Fair and Capacity scheduler. All are designed with different intentions and they all handle resources optimally. But, none of these schedulers are thermal aware and they do not do any thermal management in Hadoop cluster. To make Hadoop thermal aware, rather than developing a new thermal aware scheduler, the best strategy would be to implement a thermal module inside the existing scheduler. So, to design and implement our thermal aware module, we pick Hadoop's default FIFO scheduler, because of its simplicity of assigning tasks. It also suits our algorithm design compared to the Fair or the Capacity scheduler. Because we are just focusing on task assignment portion of the scheduler, we investigate, analyze and explain task management in Hadoop scheduler rather than block or replica management.

4.1 Hadoop FIFO scheduler

FIFO scheduler is the default Hadoop scheduler. In Hadoop the jobs are divided into tasks based on the InputSplit size and the FIFO scheduler maintains a task queue. The JobTracker is responsible for assigning a task from that queue to one of the TaskTracker that is free and ready to execute it. The job assignment and communication between the TaskTracker and the JobTracker happens primarily through the HeartBeat messages. The

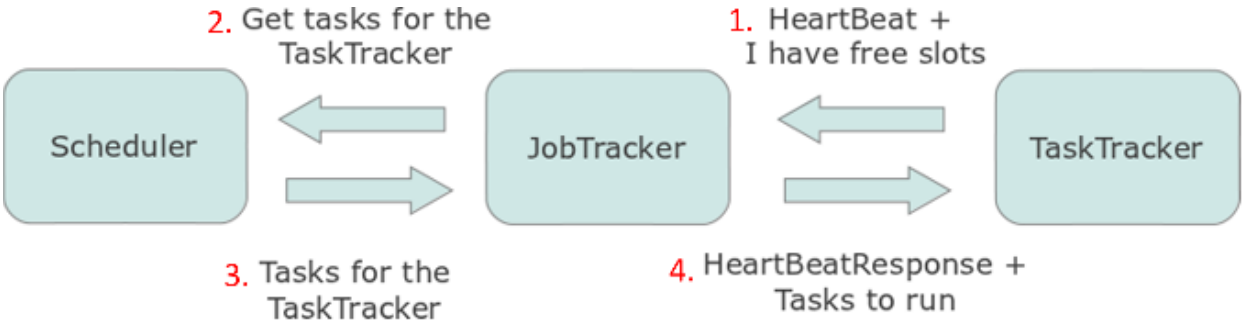

Figure 4.1: High Level TaskAssignment.

basic task assignment flow in Hadoop happens between the TaskTracker, JobTracker and scheduler as shown in the figure 4.1.

1. The TaskTracker sends a HeartBeat message to the JobTracker with the number of available slots to run Map or Reduce tasks.

2. The JobTracker receives the HeartBeat messages from the allowed TaskTrackers and requests the scheduler to assign tasks for the TaskTracker.

3. The scheduler takes the first task in the queue and gives it to JobTracker to assign it to the requesting TaskTracker.

4. The JobTracker composes the HeartBeat response along with the tasks to assign and sends it to the TaskTracker to execute.

In the essence, HeartBeat is a mechanism for TaskTrackers to announce their availability on the cluster. It is sent periodically to JobTracker to let the JobTracker know that the TaskTracker is alive. If a HeartBeat message is not received for a long duration then the JobTracker marks the TaskTracker as unhealthy and blacklists it. In addition to announcing its availability, the heartbeat protocol also includes information about the state of the TaskTracker.

- Max Map and Reduce tasks.

- Total Physical Memory and Available Physical Memory
- Total virtual memory and Available Virtual Memory
- Available space
- Map and Reduce Slot memory size
- Number of Processors
- CPU frequency, CPU time.
- Health report

Most of the fields related to the resource status of the TaskTracker are obtained form the undelying OS in TaskTracker. Because Hadoop is a Java based platform, it does not have direct access to the resources, it either uses the system commands or read system files to fill up the fields in HeartBeat.

On the other hand, the JobTracker receives the heartbeat messages and decides whether the TaskTracker is fit enough for executing a task based on it's health report and available Map/Reduce slots. It will further assign Map/Reduce tasks based on the number of free slots it has. The TakTrackers usually have 2 map and reduce slots, which means that TaskTracker can only run 2 Map or 2 Reduce tasks at a time. Generally, the number of Map/Reduce slots in a TaskTracker are configured based on the number of cores it has, one slot for each core is widely followed setting. If a node has at least 1 Map slot then the node gets a Mapper task, else it will do the Reduce task.

1. Process HeartBeat: JobTracker receives a HeartBeat from a TaskTracker with all the above fields and accepts it only if an allowed TaskTracker sent it. If the HeartBeat is duplicate then JobTracker will ignore it. Else, it will process the HeartBeat, process a response and check for the tasks to execute.

2. Get a Task: When the TaskTracker is ready to run a task, JobTracker gets the list tasks that are either a setup or clean-up task.

3. Choose a task from list: The scheduler iterates through the set up task list, and it chooses a task if the task is runnable, not running and is a failed task. It will remove a task from the list if task is scheduled, killed, completed, running or failed on this TaskTracker before.

4. Check for Flaky TaskTracker: After it obtains the Map or Reduce task, it checks if many tasks have failed on this TaskTracker before; If yes, then it does not schedule the task. Otherwise, marks the task as schedulable.

5. Available Map Slot: Once the task is marked schedulable, scheduler checks if TaskTracker has any available Map slots to run a map task.

6. Create a Map/Reduce Task: Get the number of Map slots required to run, create a Map/Reduce task by giving the job File, partition this TaskTracker owns and sets the Output directory in HDFS and creates a task entry and informs the JobTracker that this task exists. The tasks here are maintained in a FIFO queue, hence the name FIFO scheduler.

7. Assign the created Map/Reduce Task: Get the TaskTracker's total Map and Reduce Slots and get total Map and Reduce slots across the pool to calculate the load factor of Map and Reduce. Find a new Task from the FIFO queue and ensure it has all the resources and process the tasks in the order of:

    - Failed Task
    - Non-running Task
    - Speculative Task.
    - No Location information Task.

The task is assigned smartly using cache and cache levels, such that the TaskTracker has the data to be processed local to it. Scheduler tries to schedule it on local first, rack-local next, off-switch next or schedules a speculative task if other three are unavailable.

8. Launch the task: After the task is assigned, it launches the task in the TaskTracker and starts a timer. If the TaskTracker does not respond to this task for too long then it will mark the task as failed task.

9. JobTracker further checks for any jobs to be killed, cleaned up or tasks that needs to be committed. In the HeartBeat response, it checks if any restart information should be included before sending it to the TaskTracker.

All the Hadoop jobs are either CPU intensive or I/O intensive or both. In the previous chapter, we saw that the CPU and the Disk have significant contributions to a node's outlet temperature. Besides, every node may have different computation capacity as Hadoop uses commodity hardware to run the tasks. For Ex: A node may have a powerful CPU and have a moderate disk. In this case the CPU intensive tasks can be processed easily by this node rather than a disk intensive one. On the other hand, a node may be good in processing the data intensive task easily but not CPU intensive ones. This is one of the motivation for our thermal aware schedulers. So, in order to make the default Hadoop FIFO scheduler thermal aware, we propose a couple of strategies which are explained coming sections.

4.2 Design of our scheduler

To reduce the CPU or disk temperature and outlet temperature in particular, we implement the following strategies.

1. Differentiate between the CPU intensive task and Disk intensive task

2. Consider CPU and Disk utilization while scheduling

3. Maintain CPU and Disk under a threshold temperature.

AppID = 3	TaskTracker =jedi01	AvgDiskUtil=46.78	AvgCPUUtil= 74.42	TaskCounter= 142
	TaskTracker =jedi02	AvgDiskUtil=31.44	AvgCPUUtil= 33.41	TaskCounter= 78
AppID = 4	TaskTracker =jedi01	AvgDiskUtil=16.31	AvgCPUUtil= 21.87	TaskCounter= 45
	TaskTracker =jedi02	AvgDiskUtil=6.8	AvgCPUUtil= 33.15	TaskCounter= 55

4. Maintain the average CPU and disk temperature across the cluster and as a result we maintain the average outlet temperature across cluster.

One simple way to differentiate between the CPU and Disk intensive job, is by maintaining a log profile file of both CPU and I/O usage for every job that is run on the cluster. The profiled file is stored in the local storage directory (not HDFS) of the JobTracker and would include information like:

1. App ID
2. TaskTracker identification
3. Total number of tasks submitted to this TaskTracker
4. CPU utilization
5. Disk Utilization

The information in the profiled log file are not generated by the JobTracker itself, but provided by the TaskTrackers and client application. The TaskTracker sends the utilization reports of the CPU and Disk periodically while running the job. The JobTracker receives these reports from all the TaskTrackers and maintains a data structure to store the TaskTracker name or ID and average utilization of Disk and CPU for that job. The utilization may change for different jobs and makes it impossible to figure out which job had how much utilization on a TaskTracker, so we group the utilization records of a TaskTracker by the AppID. A sample record example is given in the Figure below.

It can be seen in Figure 4.2 that there are two TaskTrackers in the cluster and they have different Disk and CPU utilization reports for two different jobs. The records are grouped

by the Application name or ID. The last field in the record gives the count of number of tasks that were run on that TaskTracker.

4.2.1 Static scheduler

A good profile information log can be used to categorize an application as the CPU intensive application, Disk intensive application or both. We can set a threshold utilization beyond which, if a task tracker is utilized either on CPU or Disk, we mark it as intensive. The jobs are categorized into 4 categories for a TaskTracker

1. Not intensive.
2. Disk intensive only.
3. CPU intensive only.
4. Both CPU and Disk intensive.

For example, if the Disk and CPU utilization threshold was set at 40% and 70% respectively, for node *jedi01* the job AppID 3 would be both CPU and disk intensive, and for *jedi02* it would not be both intensive. Similarly for AppID 4 if the utilization threshold for Disk and CPU were set at 40% and 30% repectively then for the node *jedi02* JobTracker would categorize it to be CPU intensive only and none intensive for node *jedi01*.

Static scheduler would use the categorization strategy to schedule the jobs, when the same AppID job is submitted to the cluster again. The JobTracker would verify if there was a Job submitted by the same application before, by matching the application's AppID with the AppID in the job profile log. If there is a match found then it loads the utilization information of the profile log to its data structures for all the TaskTrackers. It then categorizes the job as the CPU intensive or Disk intensive for a TaskTracker. After the tasks of a job are categorized, the scheduler could simply schedule the tasks on a node, which is neither disk intensive or CPU intensive first and then it would schedule disk or CPU intensive only

jobs on the one which had less disk utilization or CPU utilization respectively. If the job was both CPU and Disk intensive for a TaskTracker, it would not schedule any tasks to it. The scheduler would not work if it does not consider the current temperature of the Tasktrackers, because it would assign the tasks to a node which is already hot or assign no jobs to a TaskTracker which is cool, but had high utilization when the job had run on this node before. For that reason, static scheduler also considers the CPU and HDD temperature at the time of scheduling a task. The static scheduler also considers the CPU and HDD temperature while scheduling, it checks if the TaskTracker is not CPU or Disk hot while scheduling tasks.

In a data center, the T_{in} is maintained to be lower than the T_{red} the Redline temperature, which is typically set at 25°C. The CPU temperature T_{CPU}, is always higher than the inlet temperature T_{in} even in idle condition and any rise in T_{CPU} would cause the T_{out} to increase and hence T_{in} would rise. So we try to maintain the CPU temperature T_{CPU} under a CPU redline temperature represented by T_{rCPU}. Similarly, the Disk temperature T_{Disk}, rise would also contribute to the rise in the the outlet temperature T_{out}, we try to maintain the T_{Disk} under the redline threshold for disk, T_{rDisk}.

$$\begin{aligned} T_{in} &<= T_{red} \\ T^i_{CPU} &<= T_{rCPU} \\ T^i_{Disk} &<= T_{rDisk} \end{aligned} \tag{4.1}$$

The application of Disk/CPU utilization and current temperature of a TaskTracker to assign a task is shown in Algorithm 1.

When the TaskTracker sends the HeartBeat message, we incorporate CPU temperature, disk temperature and utilization information in it. The TaskTracker java thread polls the OS for the temperature and utilization adds it to the HeartBeat message. The commands used for polling the temperature for CPU and disk are *sensors* and *hddtemp* respectively. Both these temperature commands give the result in degree centigrade. For the CPU and disk

Algorithm 1 Algorithm for "static scheduler", to schedule the jobs from job profile

Maintain the temperature information of all TaskTrackers
while (a TaskTracker sent a HeartBeat) **do**
  TrackerName = getTaskTrackerName()
  U_{CPU} = getCPUUtilizationFromFile(TrackerName)
  U_{Disk} = getDiskUtilizationFromFile(TrackerName)
  T_{CPU} = getCPUTemperatureFromHeartBeat(TrackerName)
  T_{Disk} = getDiskTemperatureFromHeartBeat(TrackerName)
  if ($U_{CPU} \leq U_{tCPU}$) && ($U_{Disk} \leq U_{tDisk}$) **then**
    JobTtype = EasyJob
  else if ($U_{CPU} > U_{tCPU}$) && ($U_{Disk} < U_{tDisk}$) **then**
    JobType = CPUIntensiveJob.
  else if ($U_{CPU} < U_{tCPU}$) && ($U_{Disk} > U_{tDisk}$) **then**
    JobType = DiskIntensiveJob.
  else
    JobType = CPUDiskIntensiveJob.
  end if
  if ($T_{CPU} \leq T_{rCPU}$) && ($T_{Disk} \leq T_{rDisk}$) **then**
    Schedule the job
  else if ($T_{Disk} < T_{rDisk}$) && (JobType == DiskIntensiveJob OR EasyJob) **then**
    Schedule the job
  else if ($T_{CPU} < T_{rCPU}$) && (JobType == CPUIntensiveJob OR EasyJob) **then**
    Schedule the job
  else
    Do not schedule; Process next TaskTracker.
  end if
end while

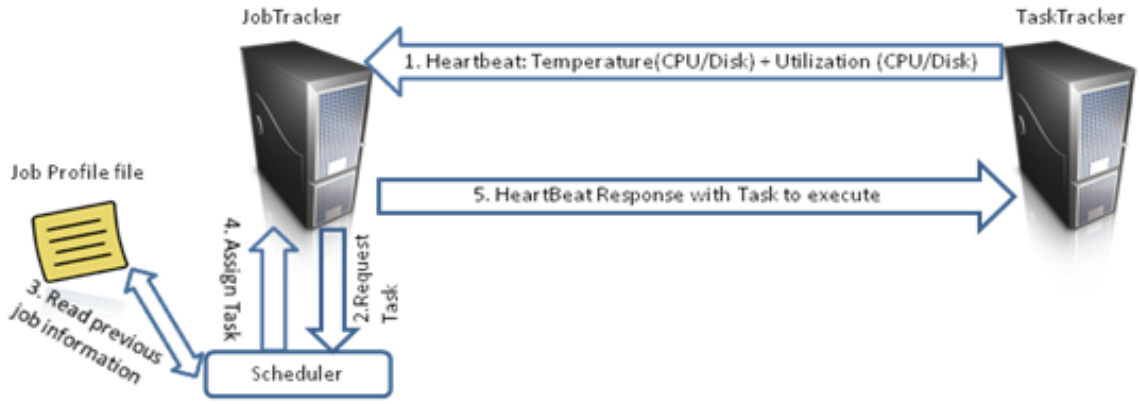

Figure 4.2: Static Scheduler communication Flow

utilization, we use the *iostat* command. The HeartBeat message interval sets the accuracy of the temperature information we have on the JobTracker.

The Figure 4.2 shows the communication between the TaskTracker and the JobTracker. Each step has a number in the scheduler flow diagram which is explained below

1. The HeartBeat messages are sent by the TaskTracker with the information of CPU, Disk temperature and utilization along with the other HeartBeat fields periodically. The Disk temperature, CPU temperature and the utilization are extracted from the TaskTracker with system commands.

2. The JobTracker receives the heartBeat message and extracts the TaskTracker status.

3. The JobTracker then extracts the message and requests the scheduler to schedule the task.

4. The scheduler meanwhile iterates through the set up task list, and it chooses a task, or removes a task if it is completed or killed. It then checks if the TaskTracker is healthy and has available map slots. It then obtains the task chosen before to create a map/reduce task. When the task is ready, the TaskTracker is assigned to execute the created task.

5. The scheduler then calls the Algorithm 1 to assign tasks based on the thermal model

    - Open the job profile file and pick up the information about the TaskTracker that has sent the HeartBeat.
    - If the Utilization is greater than the threshold utilization we set a task as data or Disk Intensive, CPU intensive or both.
    - If the temperature for both the disk and CPU are below threshold schedule it.
    - Schedule the data intensive task on a TaskTracker if the disk temperature is below the disk threshold temperature. It does not matter if the CPU is hot as the task is only disk intensive, and will increase only disk temperature.
    - Similarly, schedule the CPU intensive task on a TaskTracker if the CPU temperature is below the CPU threshold temperature and it does not matter if the disk temperature is high or low.
    - If the disk is hot and it is a disk intensive task or CPU is hot and it is a CPU intensive task or if the CPU or disk both are hot, it does not schedule any task.

6. Launch the task that is scheduled in the TaskTracker.

7. The scheduler then processes cleaned up tasks,killed tasks, and process a HeartBeat response.

The static scheduler intelligently schedules the job by keeping track of the jobs that were run on the cluster. The disk intensive tasks are only scheduled on the disk with low temperature and the CPU intensive task are scheduled only on TaskTracker with low CPU temperature. With Hadoop running either the Disk intensive or the CPU intensive the scheduler really fits to the scheme of things. The scheduler balances the temperature of the disk and the CPU across the cluster, but there is an overhead of creating and managing the job profile file. If the different applications are run then the file would be huge and also if

the cluster size is huge, then the data structure that manages would also be huge along with the profile file.

4.2.2 The Dynamic Feedback scheduler

The static scheduler always needs the file with the profiled information and it might be huge file if the data center increases the number of nodes it supports. In order to get away from the file and still reduce the temperature of the disk, CPU and outlet temperature, we implement the following strategies

1. Differentiate between the CPU intensive task and Disk intensive task
2. Consider the CPU and Disk utilization while scheduling
3. Maintain the CPU and Disk under a threshold temperature.
4. Maintain the average CPU and disk temperature across the cluster as a result we maintain the average outlet temperature across cluster.

Like static scheduler, this scheduler still gets the feedback of TaskTracker's status by the HeartBeat message. We still set the threshold redline temperature for the Disk and the CPU and try to maintain it below the threshold redline temperature. Existing thermal aware and static scheduler do not schedule any tasks, when every node in the cluster is hot, as a result the job stalls, the response time for the job increases and hence the performance drops. To manage the temperature even when all nodes are busy we need to do load balancing and keep track of the coolest nodes in the cluster to get the job going in hot conditions. We maintain data-structure to track the average cluster CPU and Disk utilization represented by U_{CPUavg} and $U_{Diskavg}$ and CPU, Disk temperature represented by T_{CPUavg}, $T_{Diskavg}$ respectively. The Dynamic scheduler is bound by constraints and one goal, to maximize Coefficient of Performance.

1. try $T^i_{CPU} < T_{rCPU}$ else $T^i_{CPU} < T_{CPUAvg}$

2. try $T^i_{Disk} < T_{rDisk}$ else $T^i_{Disk} < T_{Diskavg}$

3. $T_{Diskavg} < T_{DiskMax}$ & $T_{CPUavg} < T_{CPUMax}$

4. $T^i_{out} \leq (\sum_{i=1}^{N} T_{out})$ / N

5. Each TaskTracker is assigned only one task at a time

6. Each task is assigned to utmost one Node.

7. Minimize response time of job

The first and second constraints are that, when the disk and CPU temperature are below the threshold and make sure they operate at a temperature below the threshold temperatures. If the nodes in the cluster are already above the redline threshold, then balance the temperature by assigning the tasks to the coolest node in the cluster. The third constraint specifies that the average disk or CPU temperature is above the max temperature, then stop scheduling all the jobs as we might risk hardware failure. The fourth constraint specifies that the outlet temperature of a TaskTracker is same as the average outlet temperature of the cluster. The fifth and sixth constraints make sure that a node gets utmost 1 task and task is executed at utmost 1 node at a time. The last one tries to finish the job as soon as possible achieving optimal solution.

1. The HeartBeat messages are sent by the TaskTracker with the information of CPU, Disk temperature and utilization along with the other HeartBeat fields periodically. The Disk temperature, CPU temperature and the utilization are extracted from the TaskTracker with system commands.

2. The JobTracker receives the heartBeat message and extracts the TaskTracker status.

3. The JobTracker then extracts the message and requests the scheduler to schedule the task.

Algorithm 2 Algorithm for "Dynamic scheduler", to schedule the jobs based on real time information

Maintain the temperature information of all TaskTrackers
while (a TaskTracker sent a HeartBeat) **do**
TrackerName = getTaskTrackerName()
U_{CPU} = getCPUUtilizationFromHB()
U_{Disk} = getDiskUtilizationFromHB()
T_{CPU} = getCPUTemperatureFromHB()
T_{Disk} = getDiskTemperatureFromHB()
T_{CPUavg} = calculateAvg(T_{CPU}, n)
$T_{Diskavg}$ = calculateAvg(T_{CPU}, n)
U_{CPUavg} = calculateAvg(U_{CPU}, n)
$U_{Diskavg}$ = calculateAvg(U_{Disk},n)
if ($T_{CPU} \leq T_{rCPU}$) && ($T_{Disk} \leq T_{rDisk}$) **then**
Schedule the job
else if ($T_{Disk} \leq T_{Diskavg}$) && ($T_{CPU} \leq T_{CPUavg}$) && ($U_{CPU} \leq U_{CPUavg}$) && ($U_{Disk} \leq U_{Diskavg}$) **then**
Schedule the job as the node is cooler than the most node in clusters
else if ($T_{Disk} < T_{Diskavg}$) && ($T_{CPU} > T_{CPUavg}$) && ($U_{CPU} > U_{CPUavg}$) && ($U_{Disk} < U_{Diskavg}$) **then**
Schedule the disk intensive job
else if ($T_{Disk} > T_{Diskavg}$) && ($T_{CPU} < T_{CPUavg}$) && ($U_{CPU} < U_{CPUavg}$) && ($U_{Disk} > U_{Diskavg}$) **then**
Schedule the CPU intensive job
else
Do not schedule; Process next TaskTracker.
end if
end while

4. The scheduler meanwhile iterates through the set up task list, and it chooses a task, or removes a task if it is completed or killed. It then checks if the TaskTracker is healthy and has available map slots. It then obtains the task chosen before to create a map/reduce task. When the task is ready, the TaskTracker is assigned to execute the created task.

5. The scheduler then calls the Algorithm 2 to assign tasks based on the thermal model.

    - Extract the Disk, CPU utilization and temperatures from the HeartBeat message.
    - Calculate the average temperature and utilization for the CPU and disk across cluster.
    - If the temperature for both the disk and CPU are below threshold, then schedule it.
    - Else if the TaskTracker has both low utilization and temperature for the Disk and CPU across the cluster, schedule it.
    - Else if the CPU is hotter than the entire cluster, and disk is cooler than entire cluster, schedule disk intensive task. Scheduling disk intensive task to coolest disk in cluster keeps the disk temperature in balance across the cluster.
    - Similarly, if the Disk is hotter than the entire cluster, and CPU is cooler than entire cluster, schedule CPU intensive task. Scheduling CPU intensive task to coolest CPU in cluster keeps the CPU temperature in balance across the cluster.
    - If the disk is hotter than entire cluster or CPU is hotter than entire cluster or CPU, Disk utilization is more than entire cluster, it does not schedule any task.

6. Launch the task that is scheduled in the TaskTracker.

7. The scheduler then processes cleaned up tasks,killed tasks, and process a HeartBeat response.

To summarize, the algorithm above schedules equal tasks if all nodes are cool, if not, it then assigns tasks to the coolest nodes w.r.t disk and CPU to achieve uniform outlet profile.

Technically, the dynamic feedback scheduler is a combination of the schedulers- Uniform Outlet Profile and Uniform Task mentioned in the previous chapter. When the temperature of the TaskTrackers are below redline threshold, the tasks are distributed equally to all the TaskTrackers in the cluster (Uniform Tasks scheduler) otherwise it uses the Uniform Output Profile scheduler.

Revisitng the power consumed, we analyze the temperature saved using our dynamic scheduler. The power consumption for a node when the TaskTrackers equally share the tasks is given by:

$$P_i = G_i(\frac{C_{tot}}{N}) \tag{4.2}$$

When few TaskTrackers are already hot, we assign the jobs on coolest nodes and try to achieve thermal balancing of the outlet temperature which means that all the TaskTracker's outlet temperature should be the same (Uniform Outlet Profile scheduler). i.e $T^i_{out} = T^j_{out} = T_c$.

From equation 3.2 we have the power consumption for a node when the TaskTrackers are running to achieve thermal balancing:

$$P_i = \rho f_i c_p (T_c - T^i_{in}) \tag{4.3}$$

If a job assigned to Hadoop takes t_n seconds to finish, out of which, if it spent n seconds running same number of tasks on all TaskTrackers then it spent remaining seconds in achieving thermal balance.

$$t_n = \frac{n}{t_n} + (1 - \frac{n}{t_n}) \tag{4.4}$$

If the number of tasks performed during n seconds is represented as C_{ut}, then the number of tasks performed during the thermal balancing is $C_{uop} = C_{tot}$ - C_{ut}. Total power consumed while running the job using the Dynamic feedback scheduler will be:

$$\sum_{i=1}^{N} P_i = (\frac{n}{t_n})(\sum_{i=1}^{N} G_i C_{ut}) + (1 - \frac{n}{t_n})(\sum_{i=1}^{N} G_i (C_{tot} - C_{ut})) \tag{4.5}$$

$$\sum_{i=1}^{N} P_i = (\frac{n}{t_n})(\sum_{i=1}^{N} G_i C_{ut}) + (1 - \frac{n}{t_n})(\sum_{i=1}^{N} G_i (T_C - T_{in}^i)) \tag{4.6}$$

To find the difference in the temperature saving we express the equation in T_C

$$T_C = \frac{\sum_{i=1}^{N} P_i - (\frac{n}{t_n})(\sum_{i=1}^{N} G_i C_{ut}) + (1 - \frac{n}{t_n})(\sum_{i=1}^{N} G_i (T_{in}^i))}{(1 - \frac{n}{t_n}) \sum_{i=1}^{N} G_i} \tag{4.7}$$

$$T_C = \frac{\sum_{i=1}^{N} P_i - (\frac{n}{t_n})(\sum_{i=1}^{N} \rho f_i c_p C_{ut}) + (1 - \frac{n}{t_n})(\sum_{i=1}^{N} \rho f_i c_p (T_{in}^i))}{(1 - \frac{n}{t_n}) \sum_{i=1}^{N} \rho f_i c_p} \tag{4.8}$$

If we consider the re-circulation effect, where a_{ij} is the re-circulation coefficient when the hot outlet air from a node j affects a node i.

$$T_C = \frac{\sum_{i=1}^{N} P_i - (\frac{n}{t_n})(\sum_{i=1}^{N} a_{ij} \rho f_i c_p C_{ut}) + (1 - \frac{n}{t_n})(\sum_{i=1}^{N} a_{ij} \rho f_i c_p (T_{in}^i))}{(1 - \frac{n}{t_n}) \sum_{i=1}^{N} a_{ij} \rho f_i c_p} \tag{4.9}$$

The difference between the T_{out} and T_c would give us the outlet temperature saving that we have achieved using our scheduler over the default Hadoop FIFO scheduler.

$$\delta = \sum_{i=1}^{N} (T_{out}^i - T_c) \tag{4.10}$$

The standard deviation of the outlet temperature T_{out}^i would give us if we achieved thermal balancing or not.

$$StdDev[(T_{out})] \tag{4.11}$$

Because the outlet temeprature is a result of CPU and Disk temperature, we also consider the standard deviation of the CPU and Disk temperature as an indication of thermal balance across cluster.

4.3 Difference between Static and Dynamic scheduler

The table 4.1 summarizes and highlights key differences between the Hadoop schedulers proposed to maintain thermal balance and reduce cooling costs in a datacenter.

Static	**Dynamic**
Schedules tasks based on previously utilization information in a file	Schedules task based on utilization and temperature information at run time
Assigns CPU intensive tasks to CPU powerful nodes and Disk intensive task to Disk powerful nodes	Assigns tasks to the coolest node at any point in cluster
Does not schedule task if nodes are hot and utilization is high	schedules tasks on coolest node at any point in cluster
Does not try to maintain the uniform temperature across cluster	Maintains uniform temperatre across cluster
Active nodes are nodes that are below the temperature threshold	At least 50% are active nodes at any given time in cluster
Storage and maintainance overhead when cluster size increases or job types are different	Overhead of communication as TaskTracker report to JobTracker periodically
Works better even with smaller cluster	Works better with larger cluster
Overhead of maintaining a profile file and job information of all nodes in cluster	Overhead of communication of temperature and utilization information from TaskTracker to JobTracker

Table 4.1: Differences between Static scheduler and Dynamic scheduler

Chapter 5

Results and Interpretation

The performance of the Hadoop schedulers were measured using actual Hadoop cluster implementation. Unlike the data center clusters, we implemented a cluster of relatively small scale of 14 nodes and performed experiments and gathered the performance data of our schedulers. Similarly, due to unavailability of the large data sets, we scaled down the data set to suit the performance of the cluster size.

5.1 Experiment Setup

Using the commodity hardware we setup a Hadoop cluster composed of a JobTracker, NameNode and 13 DataNodes and TaskTrackers. All the nodes were installed with couple of real minigoose temperature sensors used to measure the inlet and outlet temperatures. Another temperature sensor at the AC duct, would give the supply temperature T_{sup} to the cluster. Apart from the real world sensors, we also measured the HDD temperature and CPU using the commands hddtemp and sensors respectively which makes use of internal sensors in a node.

5.1.1 Hardware

The commodity hardware used in the Hadoop cluster is a mixture of different configuration as it is in real data center. The configuration nodes used are given in the Table 5.1 below.

All the nodes were connected with full-duplex 10/100 Mbps Ethernet network interface cards connected to a 100 Mbps Cisco network hub.

Node	Processor and Speed	RAM	Storage
HP Xeon	4 core * 2.8 GHz	2 GB	143GB
Dell	4 core * 2.8 GHz	2 GB	143GB
Dell	1 core	1GB	143GB

Table 5.1: Node Information in Cluster

5.1.2 Software

All nodes in Hadoop cluster were running on Linux Ubuntu 10.04 operating system. For Hadoop, we used the stable version of 1.0.3 across all nodes in the cluster. To support Hadoop 1.0.3, java version of 1.6 was installed on all nodes. The nodes had password free access between them for starting the tasks and exchange of intermediate data. The nodes were also installed with the sensors, hddtemp to measure the CPU temperature and Disk temperature.

5.1.3 Cluster size and Data set

To evaluate the performance our schedulers, we change the cluster size and data set sizes. Several experiments are conducted with cluster size of 5, 10 and 14. The data sets are varied as well at sizes of 80GB, 60GB and 40GB respectively. All nodes had default map/reduce slot settings, block size and input split size. The replication factor was set to 3 in all experiments.

5.1.4 Benchmarks

The schedulers were evaluated using standard Hadoop Benchmarks. The benchmarks were diligently chosen upon the criteria of CPU utilization and Disk utilization. The benchmarks used for testing are:

- WordCount
- Distributed Grep
- PI

The first 2 application benchmarks are used for evaluating both CPU temperature and HDD temperature performance mutually. The last one is only a CPU intensive benchmark.

5.2 Results

5.2.1 Temperature Reduction

The temperature were set at different threshold values for different cluster sizes. Because CPU and Disk have different threshold for temperature, we have different values for the CPU and Disk as shown in the table 5.2. So, instead of checking the performance of different supply temperatures, we keep the supply temperature constant and evaluate at different threshold values.

CPU	Disk
32	27
35	28
38	29
41	30
44	31
47	32
	33

Table 5.2: Temperature Threshold for CPU and Disk

5.2.2 Static scheduler

In case of static scheduler, initially we run the experiment without the thermal module in the FIFO scheduler, to obtain the CPU and Disk utilization in a profile file. The stored profile file will serve as reference for our static scheduler for scheduling the jobs of same kind in future. The experiments with different threshold temeperatures, cluster sizes and application benchmarks are executed and analysed in this section.

14 Nodes, 80GB, Word count, Grep

Experiment setup included running the default Hadoop FIFO scheduler and our static scheduler for a data size of 80GB in the HDFS on a 14 node cluster. The graphs in figure5.1

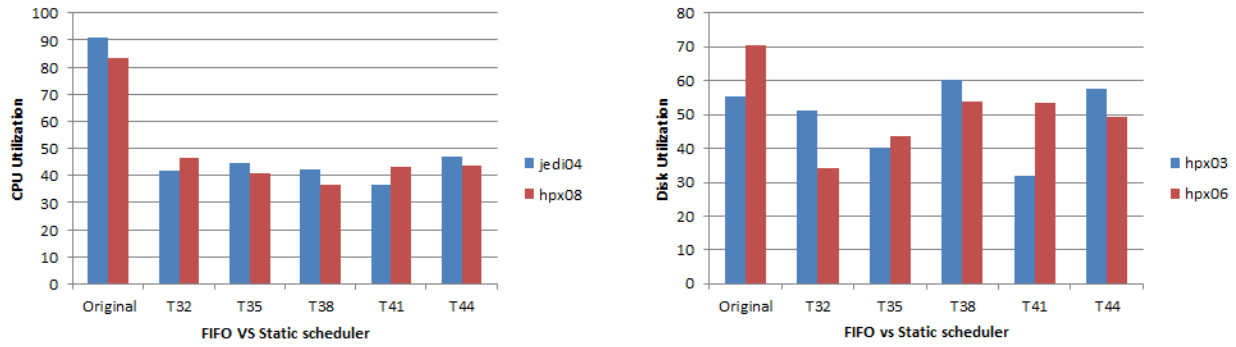

Figure 5.1: The CPU and Disk utilization

show the CPU and Disk utilization and jobs submitted to every hot node that exceeded the threshold utilization of 50% for both CPU and Disk.

The CPU and disk temperature control achieved by the static scheduler for 14 nodes, 80GB is shown in figure below.

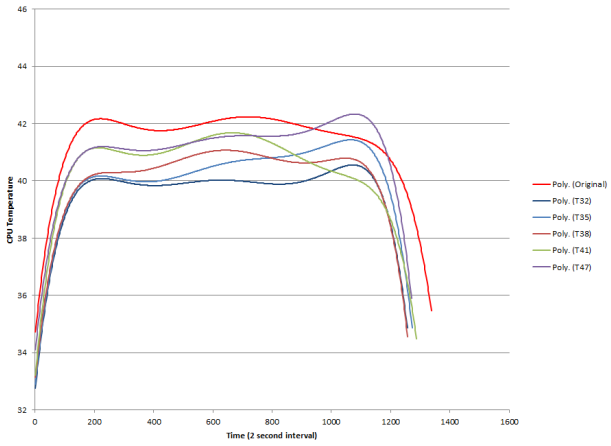

Figure 5.2: Avg. CPU temperature of 14 nodes in WordCount for Static scheduler vs FIFO scheduler

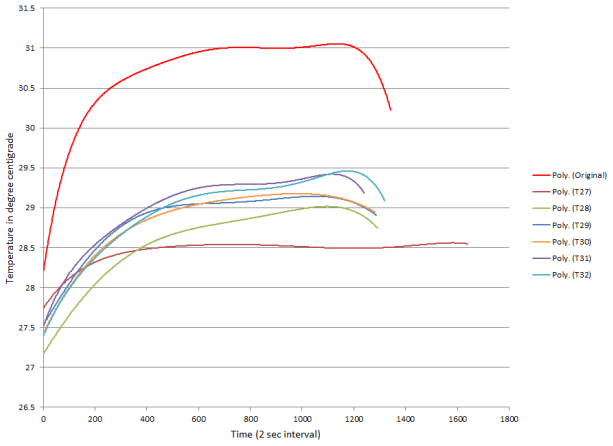

Figure 5.3: Avg. HDD temperature of 14 nodes in WordCount for Static scheduler vs FIFO scheduler

Discussion

As shown in figure 5.1, initially when Hadoop job was run with FIFO scheduler without thermal module in it, to store the utilization information of Disk and CPU in profile file. Using that profiled information, we obtain the utilization graph for CPU and Disk, which are

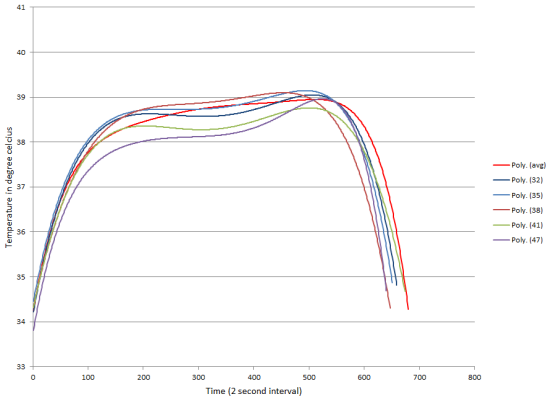

Figure 5.4: Avg. CPU temperature of 14 nodes in Grep for Static scheduler vs FIFO scheduler

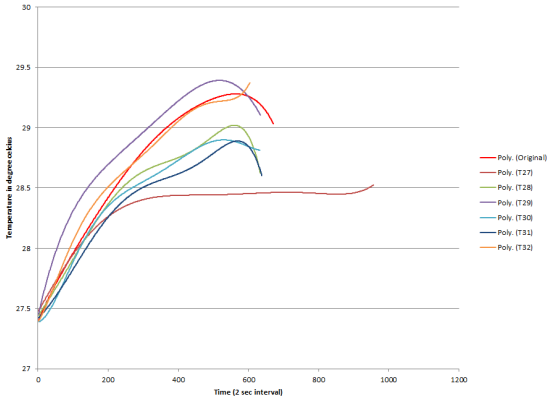

Figure 5.5: Avg. HDD temperature of 14 nodes in Grep for Static scheduler vs FIFO scheduler

represented by first 2 bars in the figure 5.1 and the rest are at different threshold temperature of CPU according using our static scheduler.

From the figure, we can see that nodes that were most utilized in FIFO scheduler are underutilized in static scheduler. The nodes *hpx03* and *hpx06* in a cluster, had high disk utilization for a job and for the same job, nodes *jedi04* and *hpx08* had high CPU utilization. Our static scheduler learned from the profiled information that the nodes with high utilization created or could create a hot-spot. Using the knowledge of utilization patterns, our static scheduler gave jobs only if the nodes were cool at the time when tasks were assigned. When the jobs are assigned to only cool nodes, the Disk and CPU utilization in hot node drops in static scheduler as compared with FIFO scheduler. An exception to this case is seen in Disk utilization at T38 and T44, where the temperature was below temperature threshold for the nodes with high utilization at the time of task assignment.

By reducing the utilization, static scheduler gave the CPU and Disk intensive tasks to CPU and Disk powerful machines only, which prevented from creating a hot-spot and achieve temperature reduction by running at relatively low utilization. As shown in figure 5.4, in static scheduler we were able to achieve best to $\sim$2°C reduction in average CPU temperature for CPU threshold temperature of 32(T32) and the average cluster temperature gradually increases as we increase the threshold temperature. Similarly, for the Disk temperature, we

were able to achieve a temperature drop of ∼2.5°C, which again gradually increases with higher threshold temperatures.

The grep application did not show as much improvement as Word count, as the utilization for CPU remained below the threshold value of 50%. When the utilization is below the threshold, the static scheduler is same as FIFO scheduler. However, for the Disk, there was a slight disk temperature drop of for ∼0.65°C lowest disk temperature threshold. In other words, grep application was not as CPU intensive or disk intensive as WordCount and hence our static scheduler categorizes all job as "easy jobs" as mentioned in chapter 4.

5 Nodes, 40GB, Word count and Grep

Experiment setup included running the default Hadoop FIFO scheduler and our dynamic scheduler for a data size of 40GB in the HDFS on a 5 node cluster.

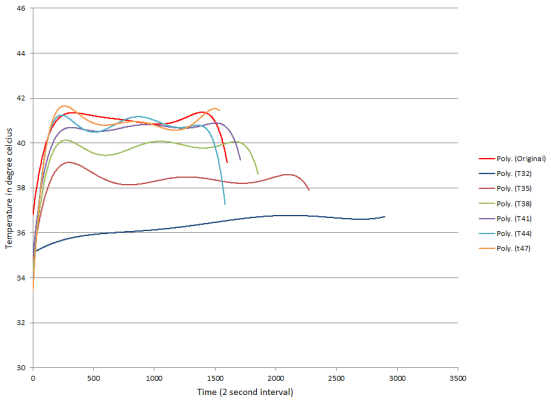

Figure 5.6: Avg. CPU temperature of 5 nodes in WordCount for Static scheduler vs FIFO scheduler

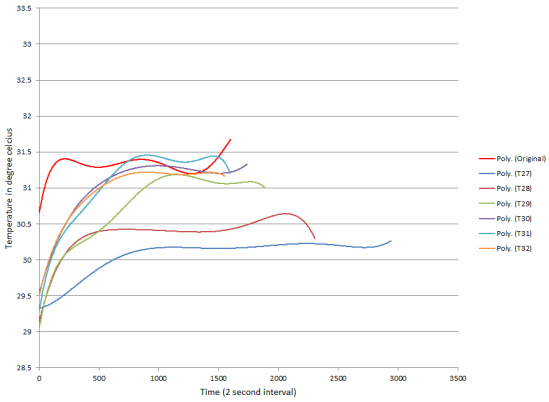

Figure 5.7: Avg. HDD temperature of 5 nodes in WordCount for Static scheduler vs FIFO scheduler

Discussion

In WordCount, the average CPU temperature across cluster showed similar behavior to 14 node WordCount as shown in 5.6. The maximum temperature difference of ∼4.7°C was again for lowest CPU threshold and FIFO scheduler. Again, as the threshold increased, the static scheduler showed same behavior as FIFO scheduler. Similarly, for Disk the maximum temperature difference of ∼1.2°C was observed for lowest disk threshold. As shown in figures

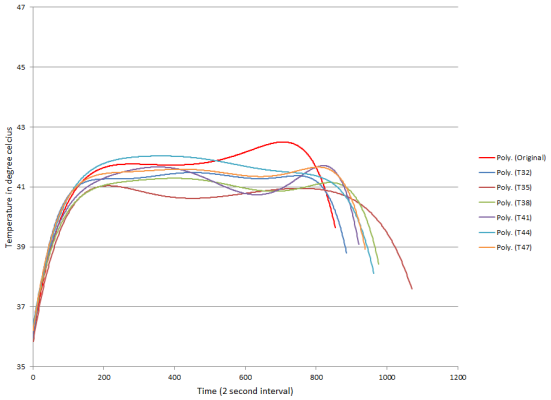

Figure 5.8: Avg. CPU temperature of 5 nodes in Grep for Static scheduler vs FIFO scheduler

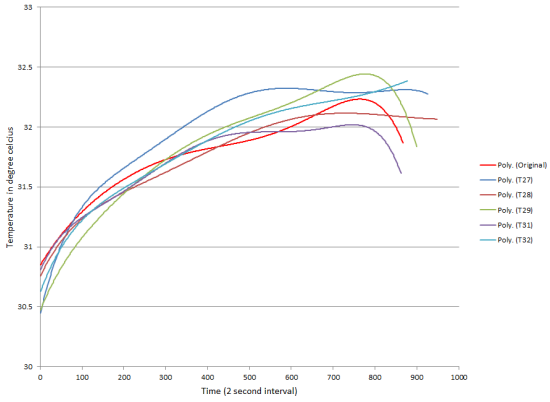

Figure 5.9: Avg. HDD temperature of 5 nodes in Grep for Static scheduler vs FIFO scheduler

5.8 and reffig:5statdugrep, the Grep job showed no improvements in 5 node cluster as well because the utilization for the disk and CPU remained below the threshold value.

5.2.3 Dynamic scheduler

Unlike static scheduler, in Dynamic Scheduler the utilization and temperature information for the Disk and CPU are given at the run-time. The scheduler tries schedule the job when the nodes are cool and maintains the average temperature across cluster by scheduling the jobs on coolest nodes leaving the hot nodes idle.

14 Nodes, 80GB for WordCount, Grep adn Pi application

Experiment setup included running the default Hadoop FIFO scheduler and our dynamic scheduler for a data size of 80GB in the HDFS on a 14 node cluster. The graphs show the average CPU and HDD temperature of the cluster and standard deviation of HDD nad CPU temperatures by running a Hadoop jobs of Word count, Grep and Pi on a data size of 80GB.

Discussion

As shown in the figure 5.10, the job was run first on a default Hadoop scheduler, followed by the dynamic scheduler runs at different CPU temperature. The maximum temperature saving of $\sim$5°C was obtained at lowest CPU temperature threshold and average CPU cluster temperature increases as the threshold increases, but the time taken to run the job decreases.

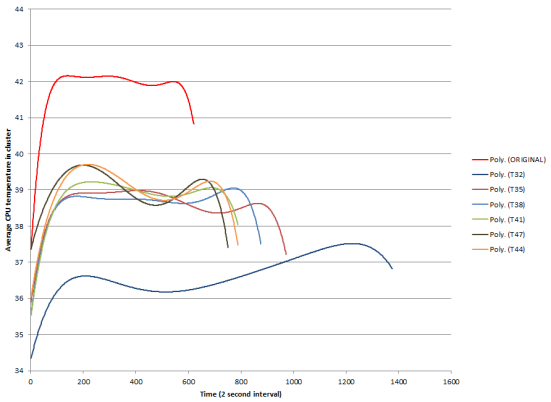

Figure 5.10: Avg. CPU temperature for 14 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

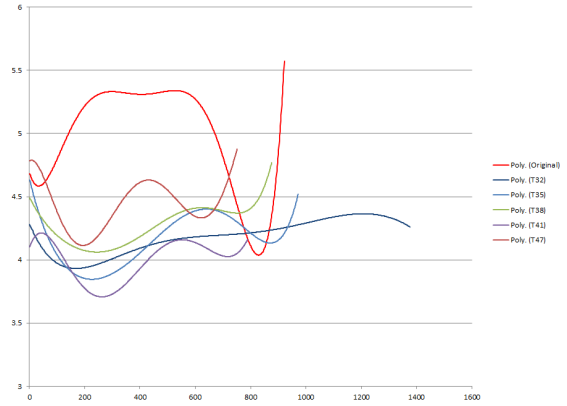

Figure 5.11: StdDev. of average CPU temperature of 14 nodes, in WordCount for Dynamic scheduler vs FIFO scheduler

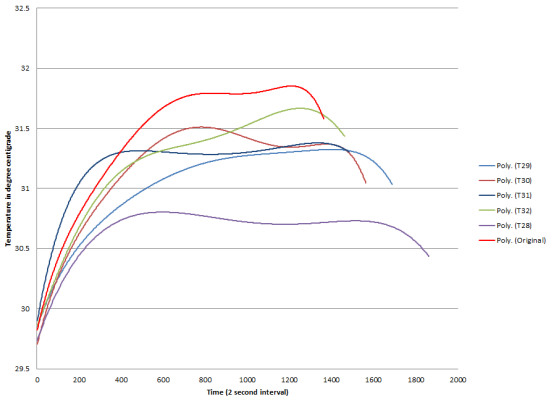

Figure 5.12: Avg. HDD temperature for 14 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

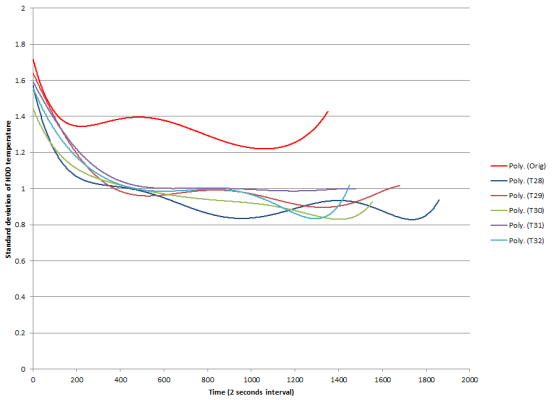

Figure 5.13: StdDev. of average HDD temperature of 14 nodes, in WordCount for Dynamic scheduler vs FIFO scheduler

At lower CPU threshold, not many nodes are available to execute the job and with that the time for execution increases. The average CPU temperatures are close to t ∼38°C thresholds after T32, because most nodes had CPU temperatures at same value and our scheduler tries to maintain the temperatures at same value. When all the nodes in cluster are around average temperature, the tasks are switched too frequently compared to the Hadoop FIFO scheduler.

Similarly, the Hard Disk temperature in Figure 5.12 showed least temperature at the lowest threshold(T28) and maximum temperature for the original FIFO scheduler. Again,

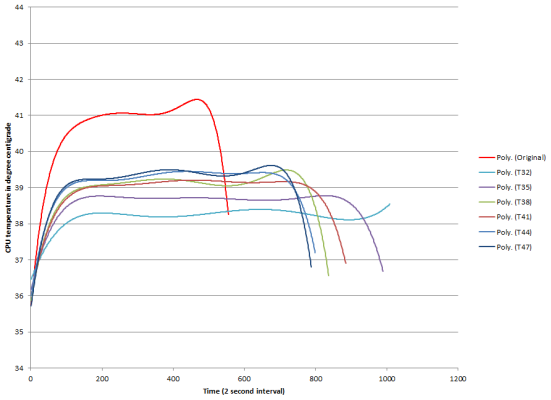

Figure 5.14: Avg. CPU temperature for 14 nodes in Grep for Dynamic scheduler vs FIFO scheduler

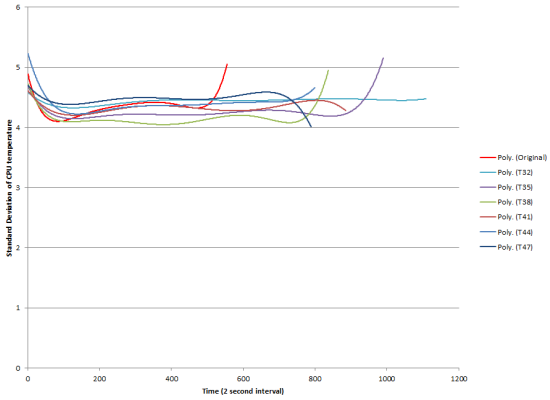

Figure 5.15: StdDev. of average CPU temperature of 14 nodes, in Grep for Dynamic scheduler vs FIFO scheduler

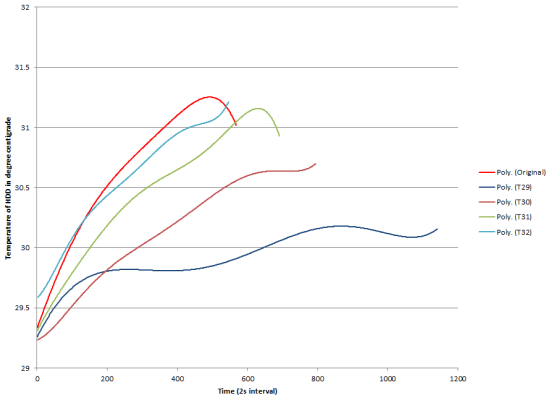

Figure 5.16: Avg. HDD temperature for 14 nodes in Grep for Dynamic scheduler vs FIFO scheduler

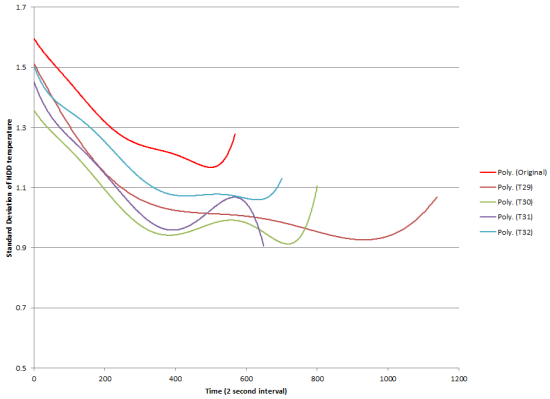

Figure 5.17: StdDev. of average HDD temperature of 14 nodes, in Grep for Dynamic scheduler vs FIFO scheduler

like the CPU temperature, HDD temperature increases with the increase in the threshold disk temperature. The maximum saving of ∼1.4°C was obtained with threshold value of T28, and least of ∼0.43°C at threshold value of T32. The time taken to finish the job was 26% more than default FIFO scheduler at T28 and drops to 4.2% at T32.

To measure the uniformity of CPU temperature in cluster, we measure the standard deviation of average CPU temperature for our dynamic scheduler at different thresholds and compare results with FIFO scheduler. In figure 5.11 and 5.11, the standard deviation of the FIFO scheduler is the highest. In contrast, the dynamic scheduler manages to keep the standard deviation of CPU and Disk temperature around the same value or lesser than

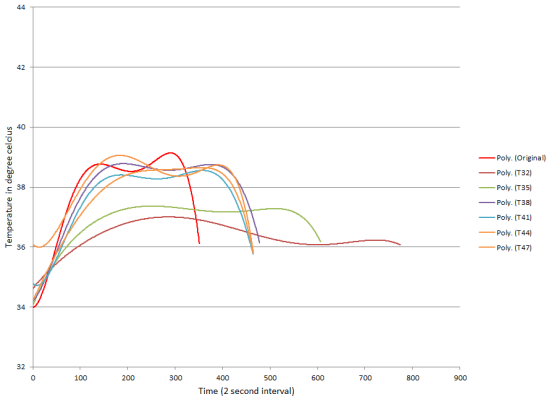

Figure 5.18: Avg. CPU temperature for 5 nodes in PI for Dynamic scheduler vs FIFO scheduler

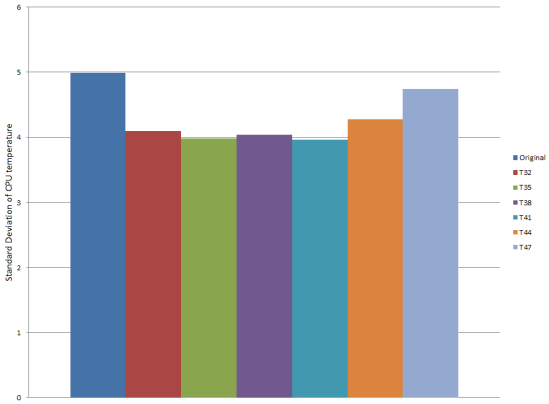

Figure 5.19: StdDev. of average CPU temperature of 5 nodes, in Pi for Dynamic scheduler vs FIFO scheduler

the FIFO scheduler, which shows that our dynamic scheduler keeps the CPU temperature uniform across the cluster by a margin of 18.5% and 23% for HDD temperature in worst cases.

In Grep application, although the temperature increased for both CPU and Disk in case of FIFO scheduler, the temperature remained uniform across the cluster because the Grep job was not computation intensive or disk intensive. From Figure 5.14 and 5.15, we can infer that using dynamic scheduler gives a maximum temperature saving of 2.8°C for almost all CPU temperature thresholds. However, for HDD (5.16 and 5.17), the HDD temperature increased with increase in temperature threshold for HDD or Disk. Using dynamic scheduler for disk intensive task in Grep application would reduce the temperature by 1°C-0.3°C.

Pi job computes value of PI, and is primarily a CPU intensive job unlike WordCount and Grep, which are also disk intensive job. In fact, pi job does not actually need any data in HDFS to work on. So in our results, we just consider the changes in CPU temperature and standard deviation of CPU temperature. The figure 5.29 shows temperature reduction of around 3°C-2°C for all thresold values of CPU temeprature. The standard deviation of CPU temperature in figure 5.19, at various threshold shows improvement of 20%.

10 Nodes, 60GB, for Word count and Grep

Experiment setup included running the default Hadoop FIFO scheduler and our dynamic scheduler for a data size of 60GB in the HDFS on a 10 node cluster. The graphs show the average CPU and HDD temperature of the cluster and standard deviation of HDD and CPU temperatures by running a Hadoop jobs of Word count, Pi and Grep on a data size of 60GB.

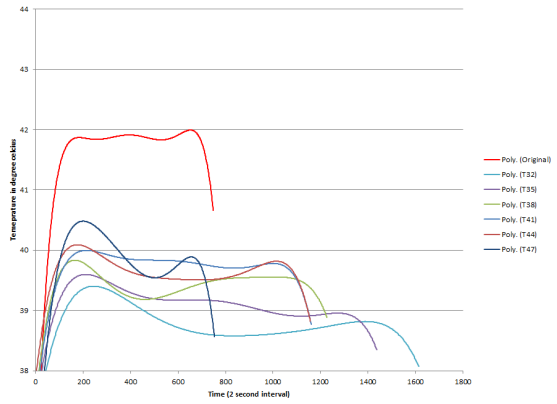

Figure 5.20: Avg. CPU temperature for 10 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

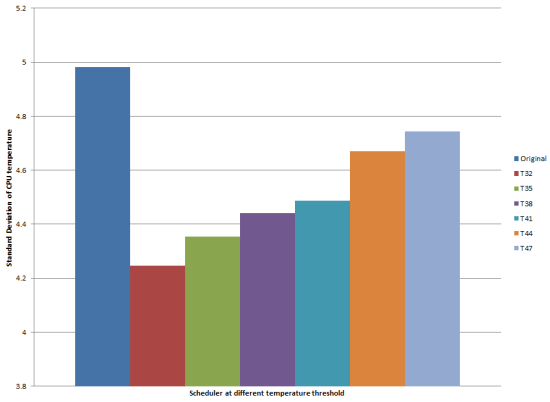

Figure 5.21: StdDev. of average CPU temperature of 14 nodes, in WordCount for Dynamic scheduler vs FIFO scheduler

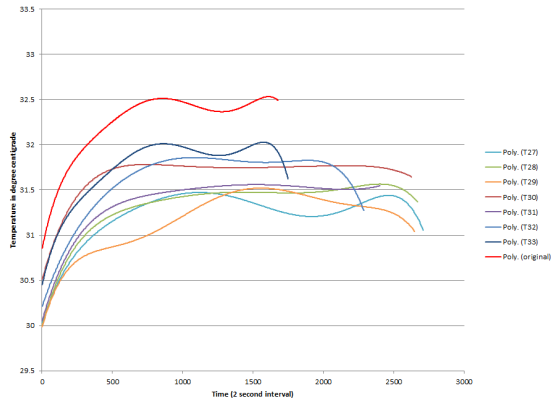

Figure 5.22: Avg. HDD temperature for 10 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

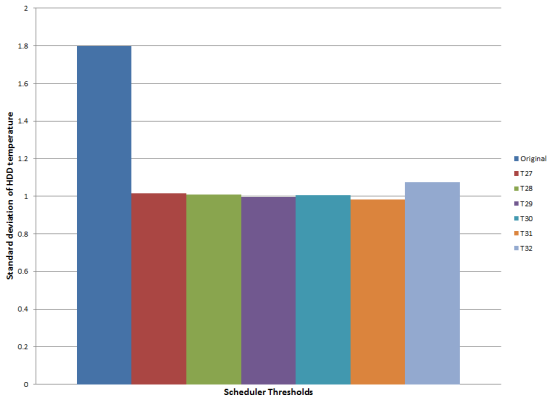

Figure 5.23: StdDev. of average HDD temperature of 14 nodes, in WordCount for Dynamic scheduler vs FIFO scheduler

Discussion

As shown in the figure 5.20 unlike 14 node cluster, temperature saving of just ∼3.5°C was obtained at lowest CPU temperature threshold and the CPU temperature increased as threshold temperature increased. Similarly, in WordCount disk intensive application, using

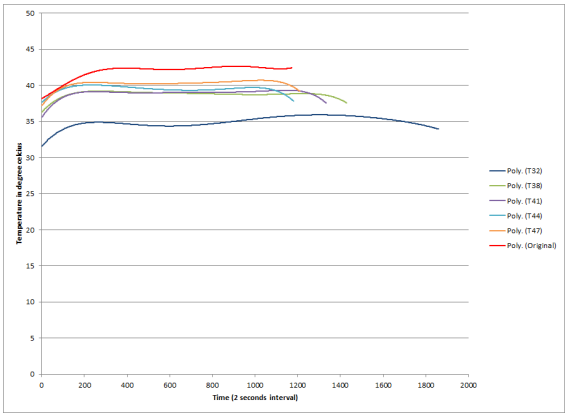

Figure 5.24: Avg. CPU temperature for 10 nodes in Grep for Dynamic scheduler vs FIFO scheduler

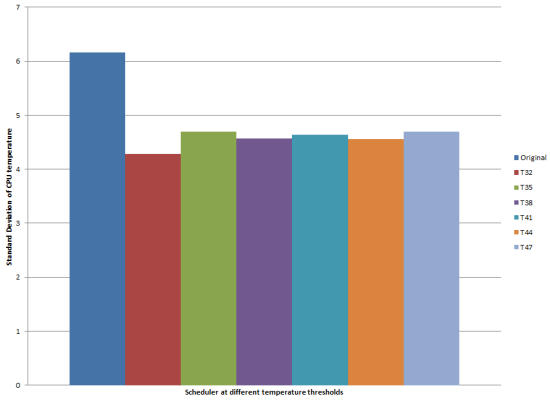

Figure 5.25: StdDev. of average CPU temperature of 10 nodes, in Grep for Dynamic scheduler vs FIFO scheduler

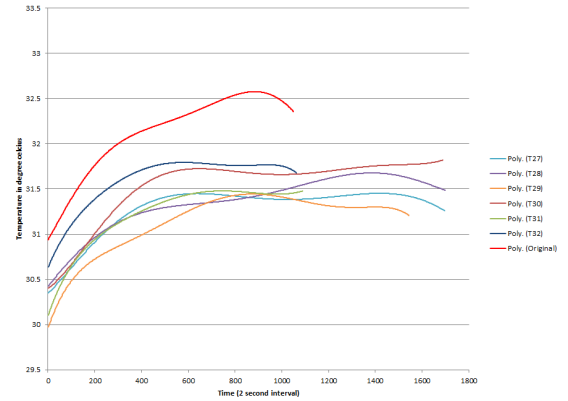

Figure 5.26: Avg. HDD temperature for 10 nodes in Grep for Dynamic scheduler vs FIFO scheduler

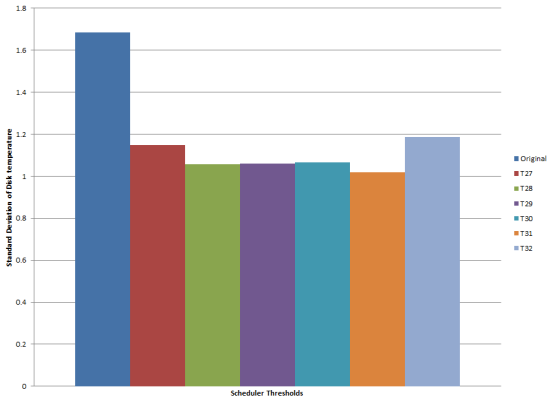

Figure 5.27: StdDev. of average HDD temperature of 10 nodes, in Grep for Dynamic scheduler vs FIFO scheduler

dynamic scheduler reduces the HDD temperature (5.22) by 1°-0.5°C. As shown in figures 5.21 and 5.23, the dynamic scheduler manages to keep the CPU and Disk temperature uniform across the cluster by a margin of 20% and 44% respectively at the lowest temperature threshold values.

Similarly for Grep, unlike 14 node cluster, average temperature saving of ∼5°C was obtained at lowest CPU temperature threshold and temperature saving is ∼3°C for the rest of threshold values(5.24). It means that in a smaller cluster, the temperature saving drops as the threshold increases, as the scheduler has less chances to schedule jobs on cooler nodes, and it rather tries to keep temperature uniform. Similarly, in Grep disk intensive application,

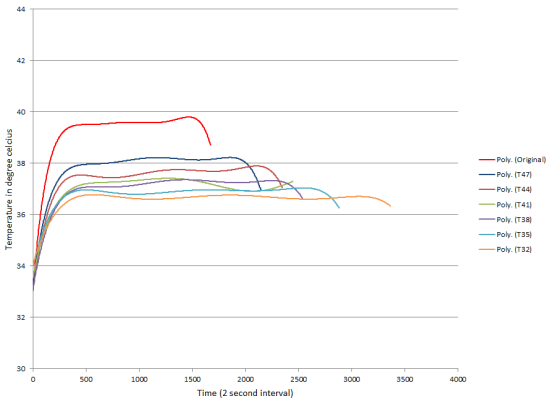

Figure 5.28: Avg. CPU temperature for 10 nodes in Pi for Dynamic scheduler vs FIFO scheduler

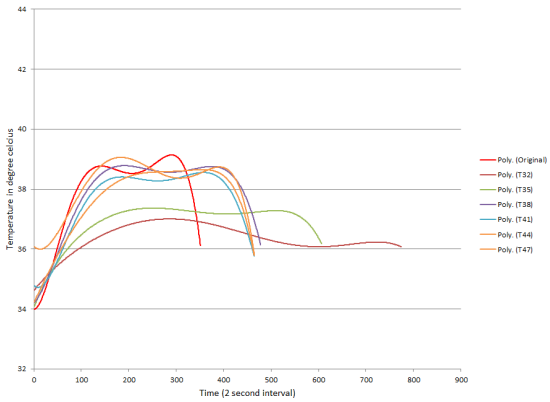

Figure 5.29: Avg. CPU temperature for 5 nodes in Pi for Dynamic scheduler vs FIFO scheduler

using dynamic scheduler reduces the HDD temperature by 0.8°C at lowest threshold(5.26). From figures 5.25 and 5.27, we can infer that the dynamic scheduler manages to keep the CPU and Disk temperature uniform across the cluster by a margin of 25% each for almost all temperature threshold values.

For Pi, the temeprature difference remained almost the same as Pi on 14 node cluster, but the temperature at higher threshold temperature increases the temperature.

5 Nodes, 40GB, for WordCount and Grep

Experiment setup included running the default Hadoop FIFO scheduler and our dynamic scheduler for a data size of 40GB in the HDFS on a 5 node cluster. The graphs show the average CPU and HDD temperature of the cluster and standard deviation of HDD and CPU temperatures by running a Hadoop jobs of WordCount, Pi and Grep on a data size of 40GB.

Discussion

When the cluster size was further reduced to 5 nodes (Figures 5.30, 5.31, 5.32, 5.33. 5.29), our dynamic scheduler executed for a longer time without significant impact on the average temperature of the cluster for both Disk and CPU, WordCount, Pi and Grep jobs. The dynamic scheduler tries to schedule the jobs on cooler nodes, but when there are fewer nodes in cluster, not every node is available for task scheduling. Even if these nodes were available, these small number of cool nodes offset the tasks of hot nodes by increasing the

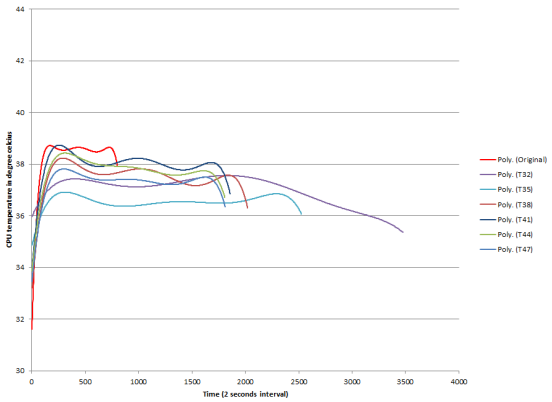

Figure 5.30: Avg. CPU temperature for 5 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

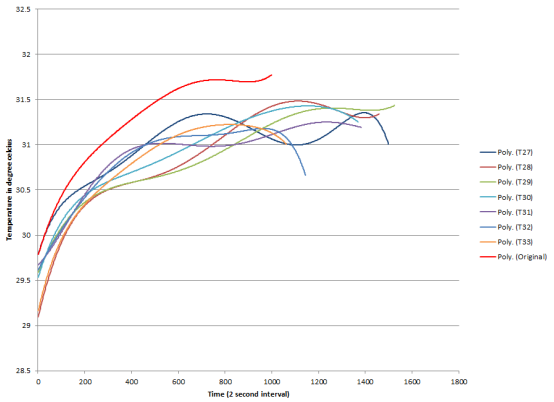

Figure 5.31: Avg. HDD temperature for 5 nodes in WordCount for Dynamic scheduler vs FIFO scheduler

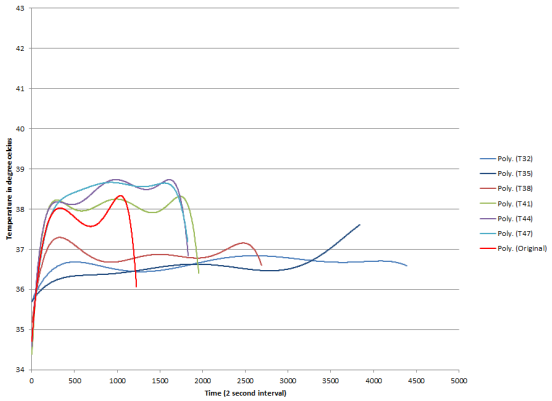

Figure 5.32: Avg. CPU temperature for 5 nodes in Grep for Dynamic scheduler vs FIFO scheduler

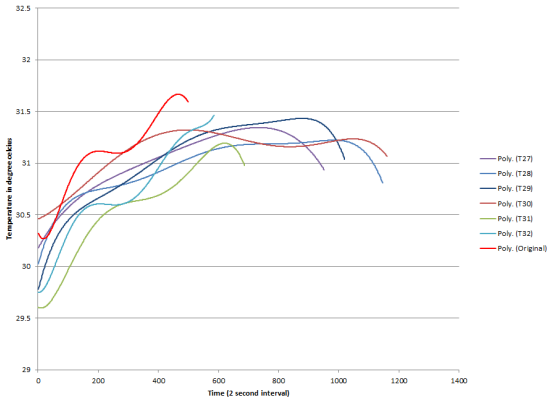

Figure 5.33: Avg. HDD temperature for 5 nodes in Grep for Dynamic scheduler vs FIFO scheduler

time take to execute a job with minimal temperature saving. It is worth observing that static scheduler was faster the dynamic scheduler and fared better in reducing the temperature for the same configuration, which means that static scheduler depends on the job type and utilization thresholds more than cluster size and dynamic scheduler depends more on cluster size.

Scalability

Our dynamic scheduler is scalable, the performance improves when the cluster size is high and drops for small clusters. Table 5.3 shows time taken and temperature saved to execute the job with node to data ratio is maintained for different cluster sizes at lowest

threshold for disk and CPU. The time taken to execute job and temperature saving is not linear as expected. The larger cluster shows better thermal management and performance compared to smaller one.

#nodes	CPU	Disk	Time
14	6.2	3.5	1.3s
10	3.4	1	1.7s
5	1	0.3	2.1s

Table 5.3: Scalability comparison at different cluster size

5.3 Overall Performance

To measure the power consumed for computation and cooling and measure saving obtained in cooling costs, we measured the inlet and outlet temperatures for original FIFO scheduler, static scheduler and dynamic scheduler. The Figure 5.34 and 5.35 shows the difference in the inlet and outlet temperature for static and dynamic scheduler, and the table 5.4 shows total power consumed, power consumed for computation and power consumed for cooling. In table 5.4 the D and C represents the disk and CPU threshold respectively. For example, D27 represents the disk threshold of 27 and C32 represents CPU threshold of 32. The sensors were used to measure the inlet and outlet temperature and from figure 5.34 and 5.35 it is clear that both static and dynamic scheduler reduces the difference between node's outlet and inlet temperatures. The temperature reduction means that T_{sup} does not have to be reduced, which increases the cooling power and cost. The power of computation and AC are calculated from the formulas in Chapter 3. The values of air density, heat of air and flow rate of air are picked from [27].

From the table 5.4 , the total power savings are calculated as shown in table 5.5.

Use of static and dynamic scheduler can save lot of power both on computation and cooling fronts. The average CPU and Disk utilization across the cluster was around 50%-60%. Since with the utilization increases the computation power and cooling power, any further increase in either of utilization will save more power and cooling costs. Although

Scheduler	**Computation Power(P_C)**	**Cooling Power (P_{AC})**	**Total Power(P)**
Original	2592.231	877.981	3470.212
dynamic D27 and C32	2198.244	753.795	2952.039
dynamic D29 and C38	2224.899	761.259	2986.159
dynamic D31 and C43	2272.578	777.047	3049.625
dynamic D33 and C45	2318.228	786.788	3105.016
dynamic D35 and C47	2338.321	795.242	3133.564
static D29 and C38	2164.083	740.050	2924.133
static D31 and C43	2226.992	750.105	2976.098
static D33 and C45	2280.925	777.455	3058.381
static D35 and C47	2306.779	772.607	3079.386

Table 5.4: Power consumption in KW for scheduler

Scheduler	**Total Power saving(in %)**
dynamic D27 and C32	14.932%
dynamic D29 and C38	13.949%
dynamic D31 and C43	12.120%
dynamic D33 and C45	10.5%
dynamic D35 and C47	9.7%
static D29 and C38	15.7%
static D31 and C43	14.23%
static D33 and C45	11.86%
static D35 and C47	11.26%

Table 5.5: Power saving for scheduler

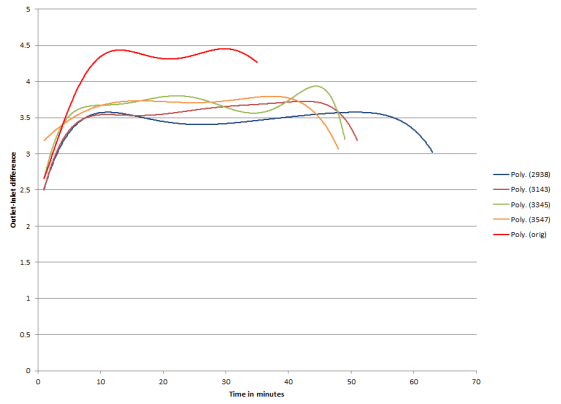

Figure 5.34: Outlet-Inlet temperatures for Static scheduler

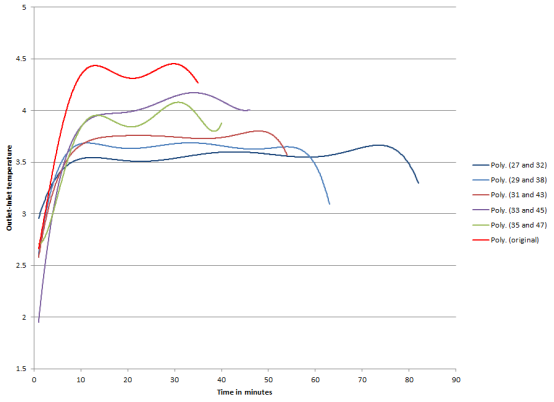

Figure 5.35: Outlet-Inlet temperatures for Dynamic scheduler

our schedulers save power, it comes at the cost of performance and execution time. It is a trade off to use which threshold values keeping execution time in mind. In fact, increase execution time does not increase the computation power as some nodes which are hot are idle and idle nodes consume less power compared to nodes running on high utilization. The execution time increased 20%-70% for static scheduler and 56%-9% for dynamic scheduler for lowest-highest threshold values. Ideally, all real time scheduling of high priority tasks should have highest threshold as job execution time is critical in that scenario and offline data processing, web analysis, low priority jobs which do not have strict time constraints should be executed in our scheduler with low threshold values for temperature.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In chapter 1 we saw how unstructured data is booming in the digital world and how Cloud Computing is being adopted by corporate world to analyze, manage and store the data. The we outlined how a distributed framework like Hadoop with MapReduce programming model is widely being used by many organizations in their Data centers to analyze big data. We then outlined the need for thermal management in data centers and described how cooling cost in data centers is increasing and becoming unaffordable. The need of thermal management in data centers is aggravated by the absence of thermal management module in Hadoop.

In chapter 2, we discussed about Hadoop architecture and how it works with it's two main components, MapReduce and HDFS. We understood that Hadoop's master-slave-client architecture and the purposes of 2 masters and slaves. We learned about HDFS- the filesystem of Hadoop used to store the data set of size peta bytes or tera bytes in form of small uniform sized blocks. The blocks are replicated to improve reliability and job localization. We also learned how MapReduce processing model then processes the data in the HDFS as ¡key,value¿ pair. The synchronization of map and reduce tasks and exchange of data were explained in this chapter.

In chapter 3, we discussed thermal model of data center in detail. The chapter explained about the power consumed for computation, cooling and various temperature that is related to the thermal model of data centers. In chapter 3, we further explained existing solutions that were proposed to reduce the cooling costs and manage heat in data center effectively.

These solutions included efficiently scheduling tasks based on thermal model in data center and hence reduce or minimize the heat dissipated by the nodes.

In chapter 4, we proposed our solutions to manage heat and reduce cooling costs in data center. We proposed 2 schedulers, a static scheduler which learns from the previous job runs to schedule the tasks on cool nodes based on utilization reports and temperature stored in profile file. The second scheduler is a dynamic scheduler which schedules the job based on current utilization and temperature of disk and CPU. We implemented both the scheduler on top of Hadoop's default FIFO scheduler and developed a thermal model using the design of our scheduler.

Finally in chapter 5, we evaluated the performance of our schedulers against Hadoop's FIFO scheduler by testing our scheduling algorithms against benchmarks like WordCount, Pi and Grep. The temperature of CPU and Disk were observed at various threshold temperatures of Disk and CPU and cluster sizes. We showed that our scheduler works better with large cluster sizes and works better with hot spots and also schedules job based on utilization pattern. In a cluster with same temperature across all nodes, the performance of the dynamic scheduler remained same as the FIFO scheduler. We successfully showed that the outlet temperature is reduced by using our scheduler and the computation power, cooling power and total power are reduced by $\tilde{1}5\%$ even for a moderate disk and CPU utilization.

6.2 Future Work

There are several lines of research arising from the work presented in this thesis. Some of interesting future researches include:

- Integrate dynamic and static scheduler: Both our schedulers work on different principles, one schedules the job on prior knowledge and another schedules the job based on current utilization. Integrating both these schedulers could use prior job knowledge and current conditions to schedule the job efficiently in data center cluster.

- Implement Re-circulation Matrix: We did not consider re-circulation effect in our thesis. But, there are schedulers which schedules the job with intention of minimizing Heat re-circulation. Our scheduler's design is flexible, it can take threshold values for each node in cluster implemented as matrix and schedule the job. Considering re-circulation matrix in our scheduler can achieve better thermal balance.

- Implement thermal module on Fair and Capacity scheduler: The thermal module is implemented currently on FIFO scheduler which has single queue to maintain tasks. The scheduler developed by Facebook and Yahoo, have multiple queues and parallel scheduling capabilities. Implementing thermal module on top of these schedulers might be interesting both on thermal management and performance fronts.

Bibliography

[1] C.B. Bash, C.D. Patel, and R.K. Sharma. Dynamic thermal management of air cooled data centers. In *Thermal and Thermomechanical Phenomena in Electronics Systems, 2006. ITHERM '06. The Tenth Intersociety Conference on*, pages 8 pp. –452, 30 2006-june 2 2006.

[2] AbdlmonemH. Beitelmal and ChandrakantD. Patel. Thermo-fluids provisioning of a high performance high density data center. *Distributed and Parallel Databases*, 21:227–238, 2007.

[3] P.E. Christian L. Belady. In the data center, power and cooling costs more than the it equipment it supports. `http://www.electronics-cooling.com/2007/02/in-the-data-center-power-and-cooling-costs-more-than-the-it-equipment-it-supports/`, 2007. [Online; created February 2007].

[4] Gartner Inc. D. W. Cearley. Cloud Computing: Key Initiative Overview. `http://www.gartner.com/it/initiatives/pdf/KeyInitiativeOverviewCloudComputing.pdf`, 2010. [Online; accessed December 2012].

[5] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.

[6] GreenGrid. Power Usage Efficiency. `www.thegreengrid.org`, 2011. [Online; created December 2011].

[7] Apache Hadoop. Hadoop Distributed File Systems (HDFS). `http://hadoop.apache.org/docs/r0.17.1/hdfs_design.html`, 2012. [Online; accessed December 2012].

[8] Brad Hedlund. Understanding Hadoop Clusters and the Network. `www-01.ibm.com/software/data/infosphere/hadoop/`. [Online; created December 2012].

[9] IBM. IBM Cloud Deinition. `http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html`, 2012. [Online; accessed December 2012].

[10] IBM. IBM Hadoop. `http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html`, 2012. [Online; accessed December 2012].

[11] IDC. IDC Press Release. `http://www.idc.com/getdoc.jsp?containerId=prUS23177411#.UMYoe-TAfTB`, 2011. [Online; created December 2011].

[12] Gartner Inc. Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond. http://www.gartner.com/it/page.jsp?id=1862714, 2011. [Online; created December 2011].

[13] Xunfei Jiang. Thermal Modeling and Data Placement for Storage Systems. Modeling of Storage systems, December 2012.

[14] Santosh Kulkarni. *"Incast-free TCP for Data Center Networks"*. PhD thesis, Auburn University, 2012.

[15] National Information Technology Laboratory. Final Version of NIST Cloud Computing Definition Published. http://www.nist.gov/itl/csd/cloud-102511.cfm, 2011. [Online; created October 25, 2011].

[16] David Linthicum. Cloud computing's role in the evolving data center. http://www.infoworld.com/d/cloud-computing/cloud-computings-role-in-the-evolving-data-center-452, 2010. [Online; created September 16, 2010].

[17] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling" cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, 2005.

[18] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[19] C.D. Patel, R. Sharma, C.E. Bash, and A. Beitelmal. Thermal considerations in cooling large scale high compute density data centers. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2002. ITHERM 2002. The Eighth Intersociety Conference on*, pages 767 – 776, 2002.

[20] Chris Preimesberger. MapR Integrates Hadoop Distro With Google Compute Engine. http://www.eweek.com/c/a/Cloud-Computing/MapR-Integrates-Hadoop-Distro-with-Google-Compute-Engine-645801/, 2012. [Online; created July 6, 2012].

[21] Bing Shi and A. Srivastava. Thermal and power-aware task scheduling for hadoop based storage centric datacenters. In *Green Computing Conference, 2010 International*, pages 73 –83, aug. 2010.

[22] Carnegie Mellon Software Engineering Institute. How Is Cloud Computing Used? http://www.sei.cmu.edu/sos/research/cloudcomputing/clouduse.cfm?location=quaternary-nav&source=591735. [Online; accessed, December 2012].

[23] Jaigak Song. Introducing a Simple PaaS Built on Hadoop YARN. http://cloud.dzone.com/articles/introducing-simple-paas-built, 2012. [Online; created July 31, 2012].

[24] Q. Tang, S. Gupta, and G. Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Cluster Computing, 2007 IEEE International Conference on*, pages 129 –138, sept. 2007.

[25] Qinghui Tang, Sandeep K. S. Gupta, and Georgios Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 129–138, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1458–1472, November 2008.

[27] Qinghui Tang, S.K.S. Gupta, D. Stanzione, and P. Cayton. Thermal-aware task scheduling to minimize energy usage of blade server based datacenters. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 195 –202, 29 2006-oct. 1 2006.

[28] Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, and Dan Chen. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, 29(3):739 – 750, 2013. ¡ce:title¿Special Section: Recent Developments in High Performance Computing and Security¡/ce:title¿.

[29] Lizhe Wang, Gregor von Laszewski, Jai Dayal, Xi He, Andrew J. Younge, and Thomas R. Furlani. Towards thermal aware workload scheduling in a data center. In *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN '09, pages 116–122, Washington, DC, USA, 2009. IEEE Computer Society.

[30] Yahoo! Hadoop Introduction. `http://developer.yahoo.com/hadoop/tutorial/module1.html`, 2012. [Online; accessed December 2012].

[31] Yahoo! Hadoop MapReduce. `http://developer.yahoo.com/hadoop/tutorial/module4.html#dataflow`, 2012. [Online; accessed December 2012].

[32] Yahoo! Hadoop MapReduce Basics. `http://developer.yahoo.com/hadoop/tutorial/module4.html#basics`, 2012. [Online; accessed December 2012].

[33] M. Yong, N. Garegrat, and S. Mohan. Towards a resource aware scheduler in hadoop. In *Proc. ICWS*, pages 102–109, 2009.

Appendices