RTAH: Resource and Thermal Aware Hadoop

by

Gautam Dudeja

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
August 2, 2014

Keywords: Data Centers, Thermal Energy, Hadoop, Map-Reduce

Approved by

Xiao Qin, Chair, Associate Professor of Computer Science Software Engineering
Cheryl Seals, Associate Professor of Computer Science Software Engineering
Alvin Lim, P Associate Professor of Computer Science 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 workload distribution, its thermal impact and overall heat distribution in the data center which leads to unstructured increase in temperature and then massive power expenditure on cooling the data center which now stands about 25% of total investment in data centers.

With the exponential increase in cooling costs of large-scale data centers, thermal management must be adequately addressed. Recent trends have discovered one of the critical reason behind the temperature rise turns out to be heat re-circulation within data center; where for a server i not only server i's workload but also its neighbor server's contribute in its temperature rise. Based on thorough investigations of Hadoop's available schedulers, we proposed a new resource and thermal aware scheduler that schedules tasks to minimize peak inlet temperature across all nodes and reduce power consumption by Air conditioning units and eventually cooling costs in data center. The proposed dynamic scheduler, schedules a job based on the current CPU, disk's utilization and number of tasks running and the feedback given by all slave nodes at run-time. This resource information gets simulated to find the respective temperature of each slave node also its neighbor's contribution. This

resource and thermal aware scheduler is 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 and TeraSort at different temperature, utilization thresholds and cluster sizes. The experimental results show that our scheduler achieve average inlet temperature reduction by 2.5C over the default FIFO scheduler that saves about 15% of cooling cost with little performance overhead.

Acknowledgments

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

I would like to express my deepest gratitude to my adviser, Dr. Xiao Qin, for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing work. He has been a true mentor and adviser in every sense of the term. I could not have imagined having a better adviser and mentor for my Masters Thesis studies. I would also like to thank Dr. Cheryl Seals and Dr. Alvin Lim for serving as members of my advisory committee, for their consistent encouragement and insightful comments.

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. In addition to it I would also like to thank all the great professor here Auburn CSSE department, for challenging me and teaching me all the things I know about computer science today. My sincere thanks to all those in my lab group Xunfei, Yuenqi, Tausif for their on going support and guidance. I owe very special thanks to Ajit Chavan my another lab mate helped me from strength to strength and being a true friend.

Last but certainly not the least I would like to thank God and my family who are miles away wishing me well every second. In addition to my family I also thank my Auburn family my friends Mohit, Sarthak, Amrit, Harshit, Vaibhav, Shubbhi, Kanika and Yasmeen for their moral support and for just being there all good and hard times. My hope is that my actions and this thesis is done in the way that would please them.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Auburn Auburn University

LoA List of Abbreviations

RTAH: Resource and Thermal Aware Hadoop

I/O: Input and Output

CPU: Central processing unit

CRAC: Computer room air conditioning

HDFS: Hadoop Distributed File System

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. Cloud computing is the next generation in computation. Maybe Clouds can save the world; possibly people can have everything they need on the cloud. Cloud computing is the next natural step in the evolution of on-demand information technology services and products. The Cloud is a metaphor for the Internet. It is a style of computing in which IT-related capabilities are provided as a service, allowing users to access technology-enabled services from the Internet (i.e., the Cloud) without knowledge of, expertise with, or control over the technology infrastructure that supports them. 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 [5]. IDC have predicted that 80% of new commercial enterprise apps will be deployed on cloud platforms [6]. Without any doubts we can say that term cloud computing phrase has become "cream of mussel" of the computing world.

Software Engineering Institute [7] studies shows the pattern that 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:

1. 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.

2. 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.

3. 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.

To provide these services and to supply the demand of computational power as a utility has increased in never heard volume and that has led to building infrastructure to handle that huge data which is being produced every second. The amount of unstructured data, also known as Big Data in Internet is growing every day. The sheer volume of data being stored today is exploding. In year 2000, 800,000 peta(1000^5) bytes data were stored in the world and by year 2012 it has grown to 8 Zetta (1000^7) bytes. Social networking and e commerce sites like Facebook, Twitter, Amazon and YouTube are generating and processing petabytes of data everyday. 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.MapReduce is the heart of Hadoop. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale out data processing solutions.

1.1 Hadoop Map-Reduce Framework

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.

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 speci

ed 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 Hadoop Distributed File System

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

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

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

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

1.3 Existing 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.

- Yahoo's 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.

- Facebook's 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.

1.4 why Thermal Management in Data Center required?

The energy consumption in data centers have seen a steady increase trend and also the power density with in the associated data center like environment. Almost half of such power consumption is been the cooling power consumption which signals that urgent and efficient solutions to reduce the energy consumption in data centers for greening the data centers. The Green Grid, a consortium of data center technology companies, is trying to make a standanrd pattern in industry to maintain and achieve energy and thermal efficiency [18]. 5.1No existing frameworks are not thermal efficient although resource and workload management is innate to achieve thermal efficiency.

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. In data centers the main contribution to power is computing and cooling power. The power usage effectiveness(PUE) [18] of a data center is total power consumed and larger PUE shows large cooling power consumption since that is the only largest non-computing power consumer. Thermal aware resource management is of prominent importance to improve cooling energy efficiency. According to US Department of Energy, average data centers PUE is around 1.7 which means around 30% - 40% is cooling cost.

Hadoop, being popularly used by enterprises that usually have average data centers, incorporating thermal awareness into it would benefit them. None of the Hadoop schedulers developed so far considers the source of temperature to handle 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.

Table 1.1: Improving energy efficiency in Data Centers

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

1.5 Contribution

In this research we propose to incorporate thermal and resource awareness in Hadoop MapReduce framework(RTAH) so as to contribute in reducing cooling power in the data centers it runs. The proposed scheme dynamically schedules tasks on the slave nodes with only intention of reducing the inlet temperature of each slave node which leads to reduction of air conditioning cost. The cooling energy depends on two factors 1) the cooling demand driven by the power distribution and the redline temperature and 2) the cooling behavior i.e. the behavior of the Computer Room Air Conditioner to meet the cooling demand. Of particular concern is the re circulation and intermixing of the hot air generated from the servers running the jobs with the cold air supplied from the CRAC. The heat re circulation depends on the data center layout and can cause hot-spots which in turn increase the cooling demand. The proposed algorithm uses slave nodes CPU and disk utilization feedback to master node. The second most critical aspect algorithm utilizes here is the heat re circulation matrix which represents the temperature effect of servers on each other. The algorithm tries to consider all heat generating sources and also its circulation affect to make the decision in order to make sure the inlet temperature of slave nodes in a rack are below a redline. Finally we evaluate and compare our Thermal Scheduler(RTAH) with the native Hadoop scheduler (FIFO) on our lab cluster using standard benchmarks and applications like word count, distributed grep, pi and tera sort at different temperature threshold and cluster sizes. The experimental results show the our scheduler achieves peak inlet temperature reduction by 2.5C and power consumption reduction by 14%. We also used governmental data for

states with data center locations and we observed the cost saving of around half a million dollar annualy for a medium size data center with 12000 cluster size.

1.6 Organization

The thesis is organized as follows Chapter 2 explains the Hadoop's architecture, HDFS and MapReduce framework in Hadop. Chapter 3 explains the problem and existing thermal solutions. Chapter 4 describes of our thermal aware scheduler. Chapter 5 analyzes the results and performances of our scheduler. Chapter 6 refers to some possible future work with conclusion to the thesis.

Chapter 2

Apache Hadoop Framework

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, exible and fault tolerant [6].

2.1 Hadoop Top 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 Top Architecture

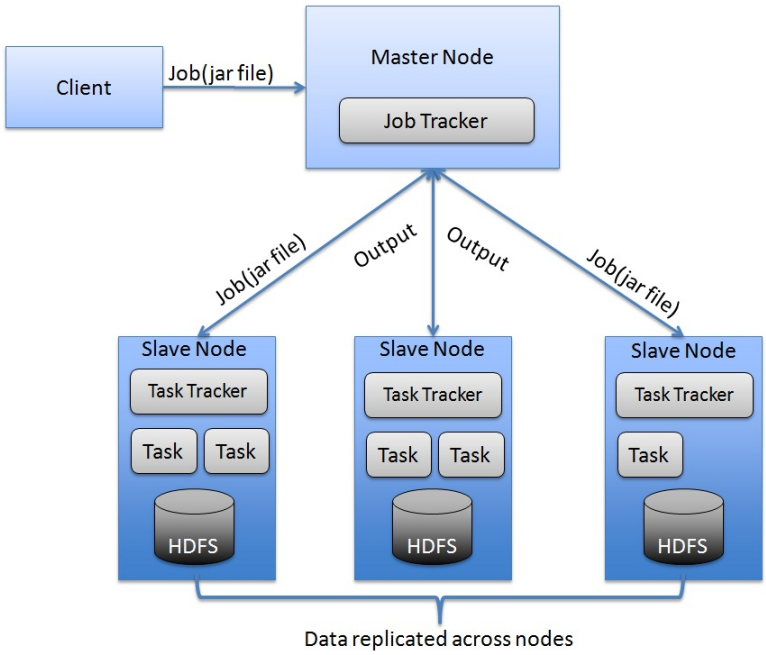


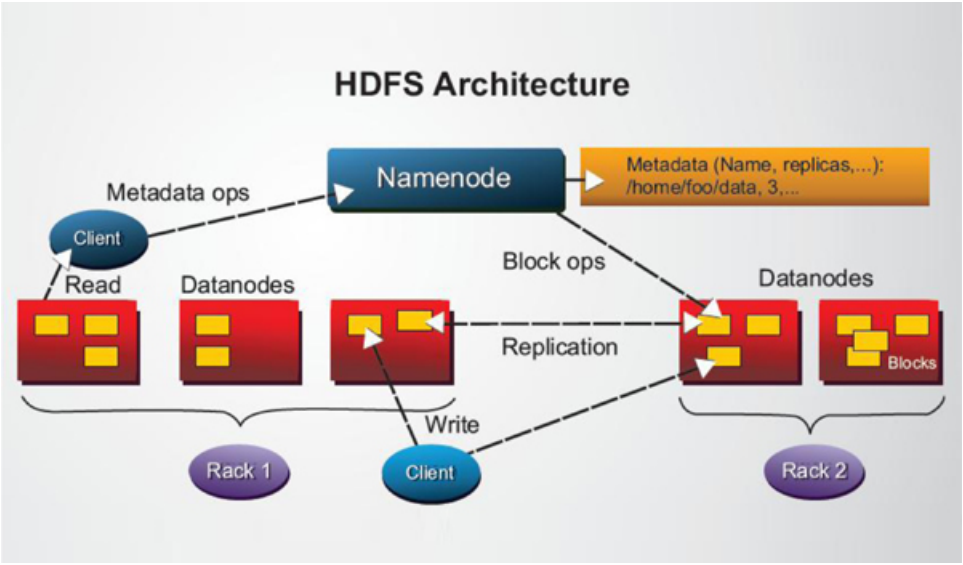
Figure 2.1 [24] shows the basic organization of the Hadoop cluster. The client machines communicates with the NameNode to add, move, manipulate, or delete files in HDFS. The NameNode in turn calls the DataNodes to store, delete or make replicas of data being added to HDFS. When the client machines want 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 file system 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. 2.2 [3]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.

Figure 2.2: HDFS Architecture



2.2.1 NameNode

Namenode is the master of HDFS that maintains and manages the blocks present on the DataNodes(slave nodes). It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. There is just one Namenode in Gen1 Hadoop which is the single point of failure in the entire Hadoop HDFS cluster. The HDFS architecture is built in such a way that the user data is never stored in the Namenode.

These are the following functions of a NameNode:

- The NameNode maintains and executes the file system namespace. If there are any modifications in the file system namespace or in its properties, this is tracked by the NameNode
- It directs the Datanodes (Slave nodes) to execute the low-level I/O operations.
- It keeps a record of how the files in HDFS are divided into blocks, in which nodes these blocks are stored and by and large the NameNode manages cluster configuration.
- It maps a file name to a set of blocks and maps a block to the DataNodes where it is located.
- It records the metadata of all the files stored in the cluster, e.g. the location, the size of the files, permissions, hierarchy, etc.
- With the help of a transactional log, that is, the EditLog, the NameNode records each and every change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- The NameNode is also responsible to take care of the replication factor of all the blocks. If there is a change in the replication factor of any of the blocks, the NameNode will record this in the EditLog.
- NameNode regularly receives a Heartbeat and a Blockreport from all the DataNodes in the cluster to make sure that the datanodes are working properly. A Block Report contains a list of all blocks on a DataNode.
- In case of a datanode failure, the Namenode chooses new datanodes for new replicas, balances disk usage and also manages the communication traffic to the datanodes.

2.2.2 DataNode

A DataNode stores data in the HDFS. A functional filesystem has more than one DataNode, with data replicated across them. On startup, a DataNode connects to the NameNode; spinning until that service comes up. It then responds to requests from the NameNode for filesystem operations. These are the following functions of a DataNode

- Datanodes perform the low-level read and write requests from the file systems clients.
- They are also responsible for creating blocks, deleting blocks and replicating the same based on the decisions taken by the NameNode.
- They regularly send a report on all the blocks present in the cluster to the NameNode.
- Datanodes also enables pipelining of data.
- They forward data to other specified DataNodes.
- Datanodes send heartbeats to the NameNode once every 3 seconds, to report the overall health of HDFS.
- The DataNode stores each block of HDFS data in separate files in its local file system.
- When the Datanodes gets started, they scan through its local file system, creates a list of all HDFS data blocks that relate to each of these local files and send a Blockreport to the NameNode.

2.2.3 Backup Node or 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 uptodate 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.

2.2.4 Replica and Block Management

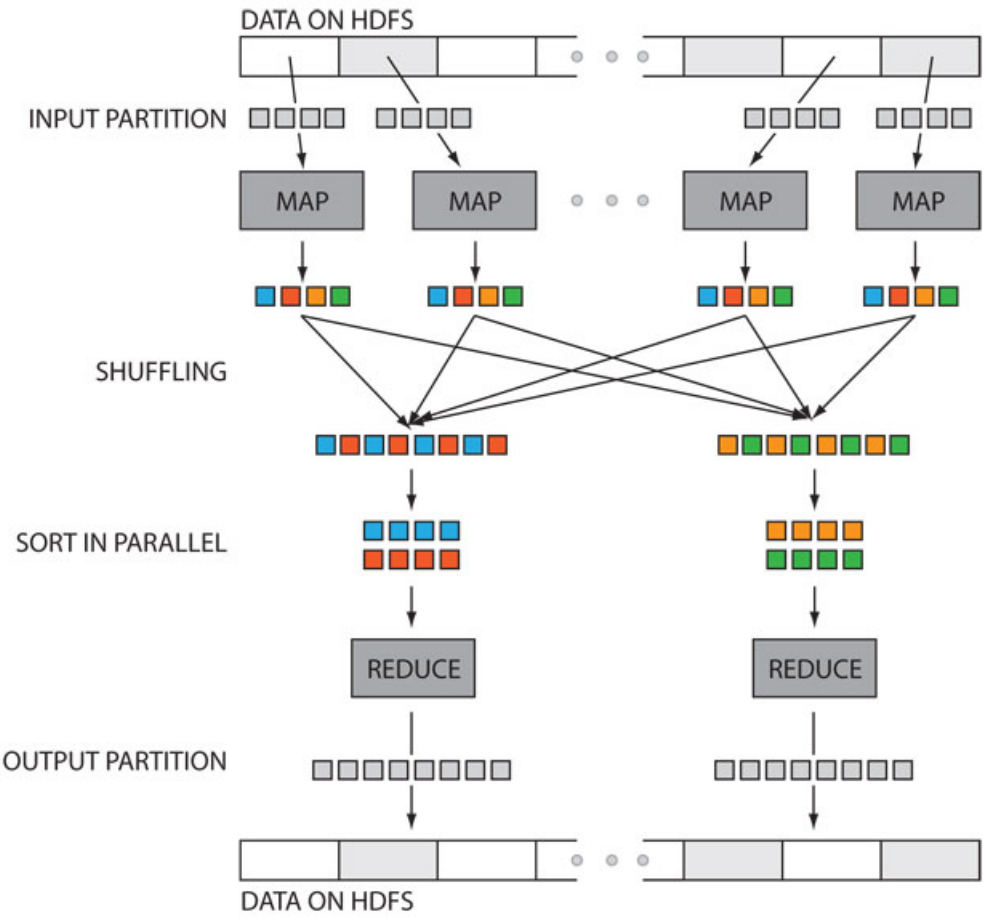
HDFS makes replicas of a block with a strategy to enhance both the performance and reliability. By default the replica count is 3, and it places the first block in the node of the writer, the second is placed in the same rack but different node and the third replica is placed in different rack. In the 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 MapRedue

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.

Figure 2.3: Map-Reduce Flow



The Figure 2.3 [4] shows the data flow of MapReduce in couple of nodes . The steps below explains the flow of the MapReduce. [4]

- Split the file: First the data in the HDFS are split up and read in InputFromat speci

ed. 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:

  - 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.

- 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 keyvalue 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.

- 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.

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

- 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 partition er determines which partition a given ¡key,value¿ pair go to. The intermediate data are sorted before they are presented to the Reducer.

- 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.

2.4 HearBeat Mechanism

As we know that once if the input file is loaded on to the Hadoop Cluster, the file is sliced into blocks, and these blocks are distributed among the cluster. Now Job Tracker and Task Tracker comes into picture. To process the data, Job Tracker assigns certain tasks to the Task Tracker. Let us think that, while the processing is going on one DataNode in the cluster is down. Now, NameNode should know that the certain DataNode is down , otherwise it cannot continue processing by using replicas. To make NameNode aware of the status(active / inactive) of DataNodes, each DataNode sends a "Heart Beat Signal" for every 10 minutes(Default). Based on this Heart Beat Signal Job Tracker assigns tasks to

the Tasks Trackers which are active. If any task tracker is not able to send the signal in the span of 10 mins, Job Tracker treats it as inactive, and checks for the ideal one to assign the task. If there are no ideal Task Trackers, Job Tracker should wait until any Task Tracker becomes ideal.

Chapter 3

DataCenter Layout and Existing Thermal Models

In this chapter we begin describing contemporary system model. Then we briefly describe general layout of data center as well a the heat circulation and cooling energy associated with a typical data center. Next we presents a review of thermal aware scheduling algorithmic solutions proposed in literature.

3.1 System Model

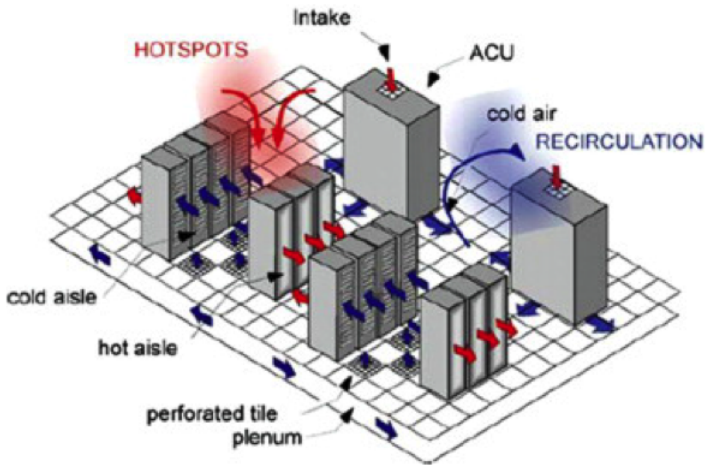
Here we start by some standard assumptions which are pretty generic in a data center like scene. we start with inferring Data center with non-uniform temperature distribution within the data center room. Believing servers are nitrogenous so they have non-uniformly contribution on the heat re-circulation in the data center room . Further, we assume the data center runs data-processing jobs using hadoop framework. Finally we assume the data-processing jobs consist of delay-tolerant batch processing jobs, such as background log analysis, that do not have strict completion time requirements; they can be delayed by a bounded amount of time. Cooling energy savings are possible by being able to temporally spread the workload, and assign it to the computing equipment's which reduce the heat re-circulation in data center room and therefore the load on the cooling systems.

3.2 Data Center Layout and Heat Distribution

0

We consider typical CRAC cooled data centers, in which the servers are arranged in chassis which are in turn arranged in racks with the racks being organized in rows. In each aisle, between two rows, the front panels or the rear panels face each other and are termed

Figure 3.1: An Overview of a data center.



cold/hot aisle respectively.The computing nodes or chassis consume power and generate heat according to the amount of power they consume. Computing Room Air-Conditioners (CRAC) provide the cooling through the perforated tiles in the floor [12]. Figure 3.1 represents a typical data center. We can see here cold aisle is towards the inlet side of the server while outlet side faces the hot aisle. Cold air is blown from CRAC which passes through perforated tiles placed in the cold aisles towards inlet sides. Ideally the hot air from hot aisle should directly go back to CRAC but here some heat gets recirculated back to inlet side of the computing nodes which leads to imbalance in thermal management and non uniformity in temperature distribution. Figure 3.1 shows that this he re circulation effect of heat creates hot spots towards inlet sides of rack, the reason being intermixing of hot and cold air. As temperature increases towards the inlet side of server now CRAC temperature should go down to bring the inlet temperature down which is also power consumption and lately has become a large chunk of data center's operational cost.

The temperature of supplied cooled air by CRAC T_{sup} should be with in the limits of red line temperature T_{red}. T_{red} depends on the client using data center. Due to re-circulation of

heat from hot aisle to cold aisle the inlet temperature vector of servers T_{in} gets represented as 3.1.

$$T_{in} = T_{sup} + \triangle T \tag{3.1}$$

here T is the temperature rise of servers due to heat re-circulation which means it heat server recieves from own outlet temperature but also from other neighbor machine. T is directly proportion to the workload that particular server processing at the time. Cooling energy consumption depends on T_{sup}. Cooling energy of CRAC can be modeled by its coefficient of performance (CoP) 3.3, which is the ratio of heat removed (i.e., computing energy) over the work required to remove the heat(i.e., cooling energy).

The 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(Equation 3.5) [11] and [20]

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

$$CoP = \frac{heat_removed}{energy_consumed_to_remove_the_heat} \tag{3.3}$$

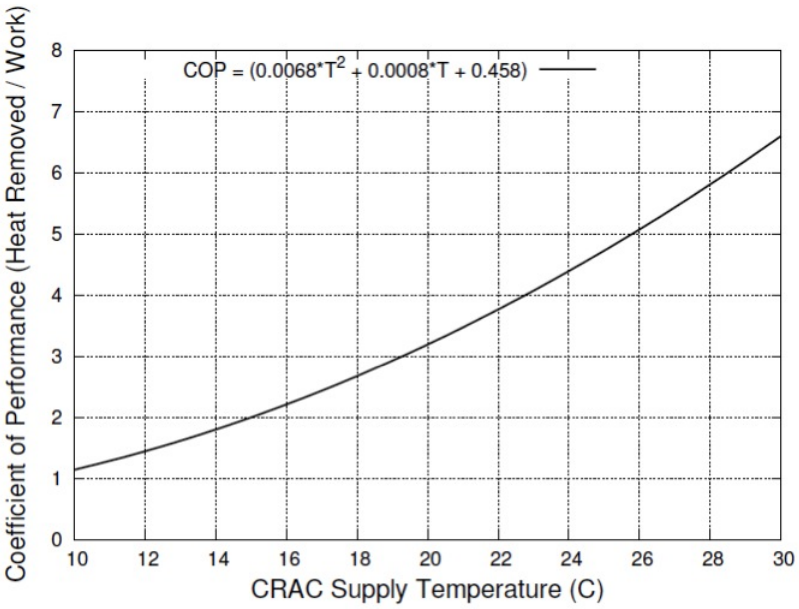
The CoP model is given by equation 3.4 where T_{sup} is supply temperature. CoP is linear(Figure 3.2) and normally increases with supplied temperature. Higher CoP shows less work to do for cooling the servers and less power consumption [12].

$$CoP = (0.068T_{sup}^2 + 0.0008T_{sup} + 0.458) \tag{3.4}$$

The power consumed by CRAC can shows in terms of computing power and COP as

$$P_{AC} = \frac{P_C}{CoP} \tag{3.5}$$

Figure 3.2: Coefficient of Performance.



In Hadoop P_C is total number of tasks that has been given to hadoop to process. These tasks boil down to basically few machine instructions, which takes different time to execute, but at cluster level, all Task Trackers are executing the similar instructions or same task. So that could be inferred as all task trackers are uniformly utilized. [22] shows that their is a linear relationship between Power Consumption of a node and CPU utilization of a machine. As their is a linear relationship between temperature and Power consumption. Using the transitive property CPU usage and temperature are also holds a linear relationship. The CPU temperature has linear model with outlet temperature of server 3.6

$$
\begin{aligned}
&P = aC_{util} + b \\
&P \alpha C_{util} \\
&T_{cpu} \alpha P \\
&T_{out} \alpha T_{cpu}
\end{aligned} \tag{3.6}
$$

Its not only CPU temperature but also disk temperature which contributes to the outlet temperature. [8] shows that an active disk contributes by almost 1.2 -1.3 degrees. They do not have a linear relation so to also count the disk contribution in the thermal model it can been seen as in two binary states as active or idle. An active disk will contribute the constant temperature rise at the server's outlet. For achieving the eventual goal of reduction in cooling cost, model minimizes the T_{inlet} and hence it minimizes the need of bringing down the T_{sup} which reduces the power consumption of CRAC.

3.3 Related Work

There is considerable work has been done to develop the efficient thermal aware scheduling algorithms for an hyper active data center. The algorithms differ in terms of workload type, optimality or complexity of the solution, the data center thermal model parameters.

- XInt-GA and XInt-SQP : XInt-GA and XInt-SQP are based on heat re-circulation coefficient for all pairs of nodes in data center, taking into consideration the data center physical layout and thermodynamics conditions.

$$D = |d_{i,j}|_{NxN} \tag{3.7}$$

  Where N is total number of cluster machines. $d_{i,j}$ is fraction of heat that flows from node j to node i. [22] reffered as HRM and can be calculated according to data center airflow simulations and data center physical layout. [23] proposed XInt-GA and X-Int-SQP to minimize the peak inlet temperatures results less cooling power. They structured the problem as a minimization of peak inlet temperature through task assignment and solve it using XInt-GA, a genetic algorithm (meta-heuristic) approach as well as XInt-SQP, a sequential quadratic approach.

- HP and Duke University in [16] and [2] built an online measurement and control techniques to enhance energy efficiency. Supply and Return Heat Index (RHI and SHI)

characterizes the energy efficiency of CRAC in data center. Using Computational Fluid Dynamic(CFD) tools which is a thermodynamic and fluid simulator, they figured out the configurations that results in hot spots and non-uniformity in heat distribution. Using those CFD simulations they dynamically distributed the workload to achieve more thermal efficiency.

- [1] gave Thermal Aware Server Provisioning(TASP) which decreases the heat generated by server provisioning. It also uses the CFD models to formulate hear re-circulation model. The authors, formulate the problem as choosing the active server set among a set of servers so as to minimize total energy. The authors solve this problem using Mixed Integer Programming (i.e., TASP-MIP) and propose a and N-approximation greedy algorithm (i.e., TASP-LRH).

- MinHR proposed by [13] is a power budgeting policy. The goal is to minimize the total amount of heat recirculated in the data center before returning to CRAC and increase power budget. It uses Hear Re-circulation Factor(HRF) as parameter which models the heat re-circulation of one chassis on all other servers (i.e., for a data centers with whose servers has homogeneous power consumption, HRF for each server is equal to the sum of the correspond row of heat recirculation matrix D). MinHR is a heuristic solution which ranks chassis based on the ratio of the HRF of each individual chassis to the sum of all HRFs, and assign tasks to the chassis with lowest HRF ratio.

- Inverse-temperature: This algorithm proposed by [19]where imbalances in temperature are resolved heuristically by redistributing workload inversely proportional to the chassis outlet temperature. Thermal load balancing is achieved by providing cooling inlet air for each rack below redline temperature, maintain uniformity in inlet temperature and also dynamically responding to thermal emergencies that cause uneven temperature.

We can see here all the above algorithms introduce a solution to choose thermal balance in data center. All solutions differ with their approach of handling the critical issue here of thermal efficiency. None of them are lithe to use them in real time framework like hadoop to really garner the solutions proposed in primarily in simulated territory. In next chapter we propose the dynamic scheduler implemented on real time distributed framework like Hadoop which uses XInt-GA like approach of minimizing the peak inlet temperature by also considering the Heat Circulation Matrix generated by CFD one time simulations of data center's physical layout.

Chapter 4

RTAH:Resource and Thermal Aware Scheduler

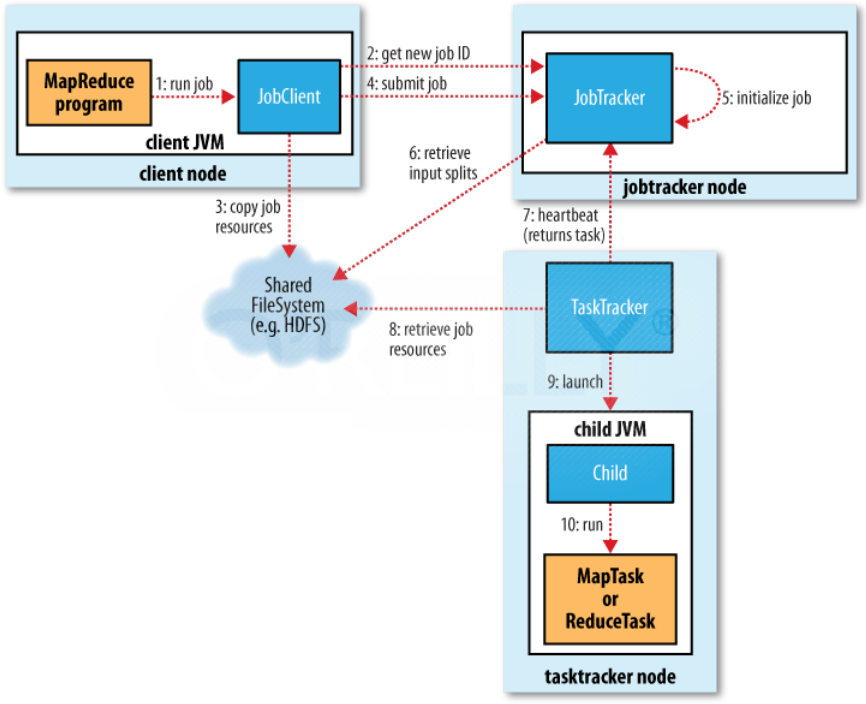
Scheduler in Hadoop primarily designed to share Map Reduce cluster for several jobs. Over time it has grown to support hierarchical scheduling, preemption and multiple ways of organizing and weighing jobs. Hadoop schedulers are like a pluggable interface and its possible to switch between them. By default Hadoop had three schedulers FIFO, Fair and Capacity scheduler. All these schedulers are structured to handle resources optimally but none of them really holds any thermal management aspect to them. RTAH here is been build on top of FIFO scheduler which is default scheduler Hadoop uses for processing the jobs. we chose FIFO because its already an optimized scheduler and adding thermal aware aspect to them will certainly be the best decision.

4.1 FIFO scheduler

In Hadoop jobs are divided into tasks based on the split size of input data. Those tasks are then stored into a task queue. Job Tracker assigns tasks from the task queue to one of the task tracker in cluster which is healthy and free to perform the task. The interface between Job Tracker and Task Tracker is the mechanism called HeartBeat which we introduced in chapter 1.

- After Job client submits the job to the cluster. As figure 4.1 from [15] shows it puts it into an internal queue from where the job scheduler will pick in up and initialize it. Initialization involves creating a task queue which is nothing but total data size divided by block size being configured that is total number of splits.

Figure 4.1: An anatomy of typical FIFO scheduler.



- After Splits being calculated Job Client creates map tasks for each split and each task being given an id.

- Now Task trackers run a simple loop that periodically (where period could be customized) sends heartbeat method calls to job tracker. The heartbeat message is the means of communication between the Task Tracker and the Job Tracker. The key fields which this message contains maximum Map and Reduce tasks,total physical and virtual memory, frequency of CPU and CPU time, responseid, state of Task Tracker health. The heartbeat message is sent periodically be the Task Tracker to the Job Tracker and if the Job Tracker does not get a heartbeat message from the Task Tracker for an interval of time, then it labels that node as unhealthy and allocates the tasks given to that Task Tracker to other healthy Task Trackers.

- In acknowledgement of heartbeat Job Tracker will further assign Map/Reduce tasks based on the number of free slots it has. The Task Trackers usually have 2 map and

reduce slots, which means that Task Tracker can only run 2 Map or 2 Reduce tasks at a time. Generally, the number of Map/Reduce slots in a Task Tracker 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.

- Before processing heartbeat Job Tracker calls for Task Tracker profiling on the basis on its previous heartbeat information.
  - Process HeartBeat: Job Tracker receives a HeartBeat from a Task Tracker with all the above mentioned data and accepts it only if an allowed Task Tracker sent it. If the HeartBeat is duplicate then Job Tracker will ignore it. Else, it will process the HeartBeat, process a response and check for the tasks to execute.
  - When the Task Tracker is ready to run a task, Job Tracker gets the list tasks that are either a setup or clean-up task.
  - 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 Task Tracker before.
  - Check for Flaky Task Tracker: After it obtains the Map or Reduce task, it checks if many tasks have failed on this Task Tracker before; If yes, then it does not schedule the task. Otherwise, marks the task as schedulable.
  - Assign the created Map/Reduce Task: Get the Task Tracker'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

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

Job Tracker 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 Task Tracker.

4.2 RTAH Design

The most important part of RTAH is to have required information about task trackers health and its current Disk and CPU utilization to be communicated to Job Tracker. The given information a Temperature Simulator consumes to calculate Temperature information Scheduler needs in order to keep the Peak Inlet Temperature below Redline.

4.2.1 HeartBeat Mechanism Modification

Our one of the main design goal was to build a propagation system between Task Tracker and Job Tracker to get the required information of task tracker to scheduler in real time. There are few options from available mechanisms which already exist in Hadoop such as Metrics, Counters and HeartBeat. Metrics are not part of Hadoop's internal MapReduce interface in fact it runs as an external process. We chose to use Heartbeat because of its light weight nature. There are 2 Heartbeat interfaces in Hadoop cluster one is between Job Tracker and Task Tracker while another between DataNode and NameNode. For our schedulers prospective we used the Mapreduce interface.

At Task Tracker it maintains an instance of a class called ResourceStatus which composed of all resource information about the task trackers resources such as Virtual memory, Physical Memory, number of map/reduce slots. We modified the Resource status class and added few more methods to calculate the CPU Utilization, Disk Utilization, CPU core temperature and Disk temperature. Since Hadoop is a java based framework and running on

JVM it can't access the resource of any machine. So we used some Linux Resource Calculator plugins such as /proc/stat which provides the CPU and Disk information. For Disk Temperature we used another Linux application called hddtemp to extract the Disk Temperature.

The advantage of using the heartbeat to communicate the data from Task Tracker to Job Tracker is to leverage an existing lightweight mechanism that is very well suited for real time data propagation. We also recognized that the data is real-time upto the period of the heartbeat which is minimum 3 seconds.

JobTracker keeps a HashMap of Task Tracker Name to last TaskTrackerStatus received from that Task Tracker. The ResourceStatus instances can be accessed through TaskTrackerStatus.getResourceStatus().

4.2.2 Thermal Simulator

We described the plan of the model used as well as the basic components necessary for the model. In this section, we will present the assumptions and the notations we used in the model. Following are the assumptions :

1. Initial temperature is always consistent which is room temperature throughout the data center.
2. The air flow is static in all parts of the data center.
3. Supplied temperature strength is linearly proportional to the distance from the vent.
4. disk is assumed to be in two states either active or idle.
5. we assume in active state disk contributes constant rise in outlet temperature.

At Job Tracker a Temperature simulator class gathers CPU and Disk Utilization from the Hash structure maintained by Job Tracker. For a server i the outlet temperature is sum of $T_{initial}$ and T_{rise}. Here T_{rise} is linearly related to the CPU Utilization of server i as equation

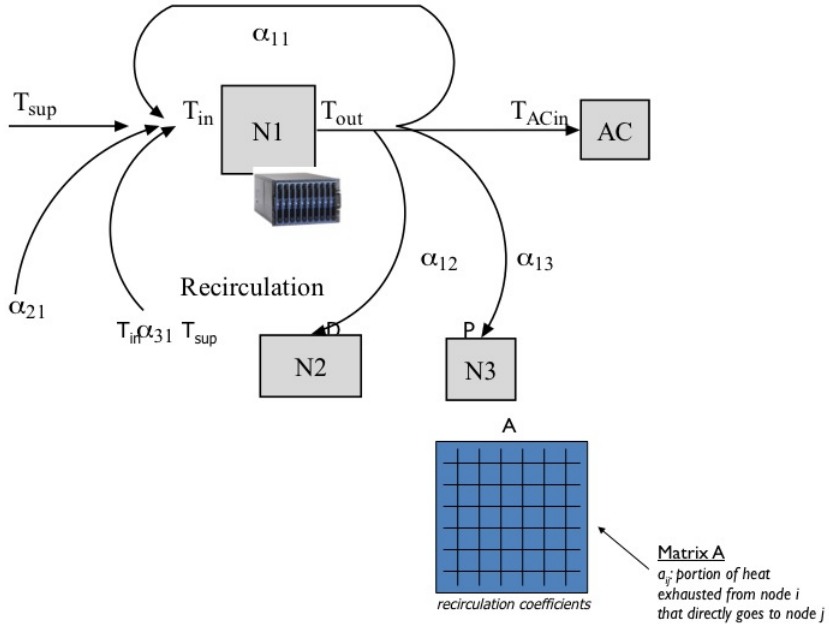
Table 4.1: Model Notation

Variables	Description
i	Number of Server Node
Q	Heat generated (J)
p	Density of air (kg/m^3)
f	Flow rate (m^3/s)
T_{sup}	Air temperature as supplied from cooling unit
Anxn	Heat Re-circulation/ cross-interference matrix
T_{out}	Outlet Temperature (c)
T_{in}	Inlet Temperature (c)
T	Change/Rise in temperature (c)
T_{red}	Red Line/Threshold temperature
P_c	Computational Power
R	Ratio of distance
$d_{i,j}$	Cross Interference matrix
d_i	Distance of the server from AC vent (m)
d	Height of room (m)
$T_{initial}$	Room Initial Temperature (c)
K_{disk}	constant temperature rise by disk
$T_{in}^{possilble}$	Possible Inlet temperature if job assigned
C_{util}	CPU utilization
D_{util}	Disk Utilization
P_{AC}	cooling cost/Power consumption by AC unit
COP	Coefficient of performance

4.2 where K_{disk} is a constant contribution of an active disk to the outlet temperatureT_{out}. We know the Inlet temperature of a server node(Task Tracker) gets affected by CRAC temperature T_{sup} and also by outlet temperature of itself and also the contribution by other server nodes because of heat re-circulation effect. Equation 4.1 shows the actual representation of Inlet Temperature of a server at any given point of time. In the article [21], they define T_{in} as dependent on T_s and a vector which models the exact strength of T_s at each height.

Here R represents the factor of height of server from CRAC this gets calculated using ratio of server's height from CRAC to ceiling height from CRAC. This in lines of the fact that the higher the server's placement from CRAC which is at floor the less cooling effect it will receive from cooling device. Inequation 4.1 A is a heat re-circulation or cross-interference

Figure 4.2: Cross-Interference between neighbor servers



vector which sums up the temperature rise of server node i because of its neighbors outlet temperature sitting at placement for say at i+1, i+2 or i-1.

Heat Re-circulation vector or matrix gets calculated using thorough evaluation of physical layout of data center using CFD tools as proposed in [17] and [13]. In figure 4.2 A is a nxn matrix where n is total number of cluster nodes in the cluster. each element $d_{i,j}$ from the matrix represents the fraction contribution of heat from server i's outlet to j's inlet temperature.

$$R = 1 - \frac{d^i}{d}$$
$$T_{in}{}^i = T^i_{initial} - RT_{sup}{}^i + AT_{out} \tag{4.1}$$

This equation can be reorganized to solve for outlet temperature.

$$T_{rise}{}^{i} = K_{disk}{}^{i} + aC_{util}{}^{i} + b$$
$$T_{out}{}^{i} = T^{i}_{initial} + T_{rise} \tag{4.2}$$

The main reason for T_{out} to rise is the heat transfer in the system from inlet side to outlet side. The PC components such as CPU chip and disks heats up on processing the workload and with air heat transfer it leads to rise in T_{out}.

4.3 Implementation

The algorithm in goal of minimizing the inlet temperature uses some input parameters. which are listed here below:

1. A[][] : a cross interference matrix/heat re-circulation matrix
2. CPU and Disk utilization from heartbeat message
3. height[] vector which keeps the height information of each server with respect to CRAC at floor.
4. Currently running map tasks on task tracker i, also supplied via heartbeat.

Now as previously stated when the Task Tracker sends the HeartBeat message, we incorporate CPU usage and disk utilization information in it. The Task Tracker java thread polls the OS for the temperature and utilization adds it to the HeartBeat message. For the disk and CPU utilization we use *iostat* command. The HeartBeat message interval sets the accuracy of the real time information we have on the Job Tracker. Now there are few assumptions which were considered for algorithm such as:

1. Try to keep T^{i}_{in} below T_{red}
2. Each Task Tracker will be assigned one task at a time

3. All map tasks in a job are identical in terms of their resource requirement because they all process the same operation.

4. All map tasks are processing the same input block size

5. Each task will be assigned to utmost one Task Tracker

6. Above constraints meant that Temperature rise per map task will also be same for a server

7. Heat re-circulation effect is more profound to servers belong to same rack.

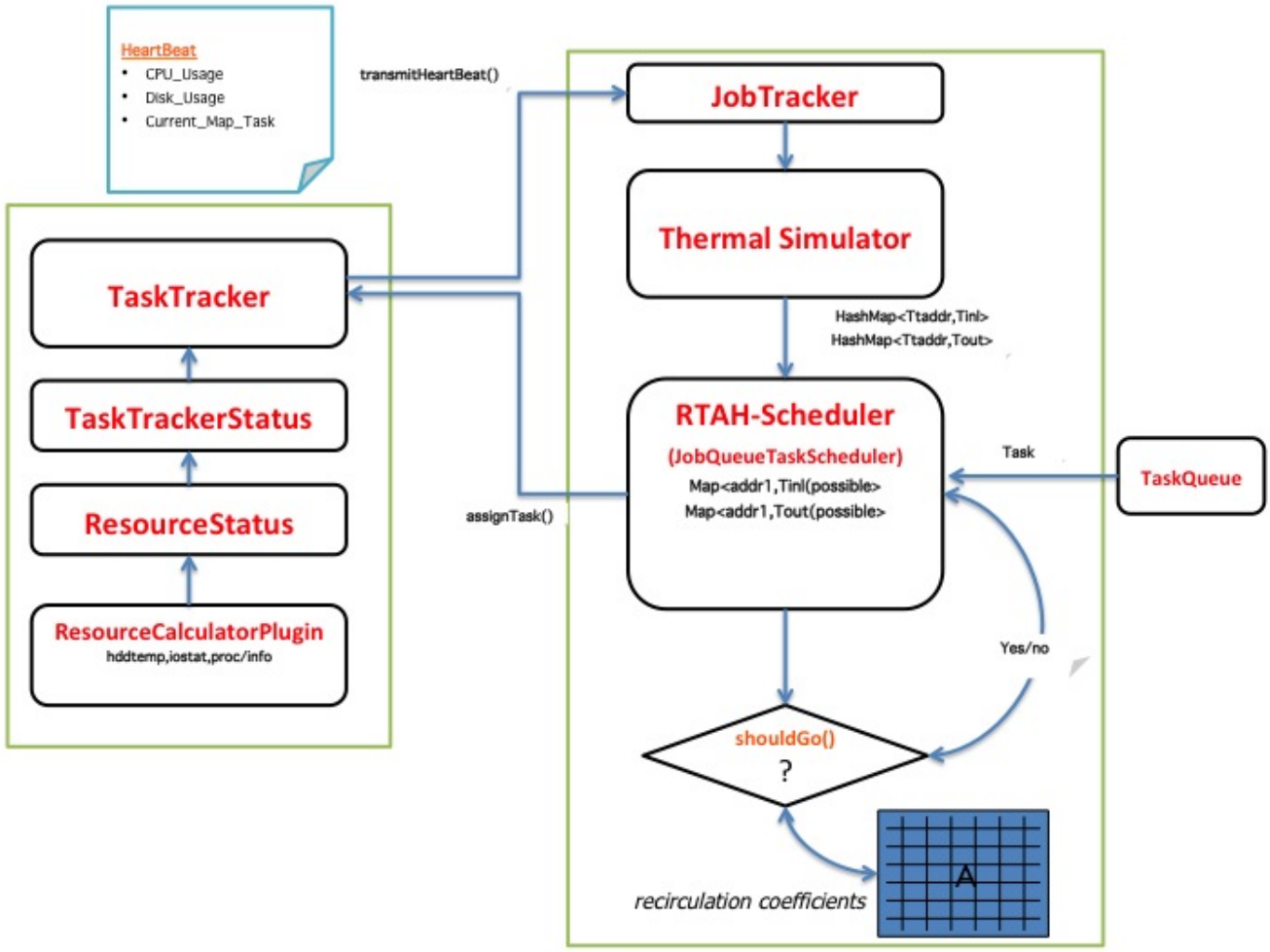
The first assumption here is decision making condition in scheduling to keep inlet temperature below redline and this way no need to decrease the CRAC temperature T_{sup}. The FIFO scheduler by default assigns one map task to each task tracker whenever task tracker asks for a task from the task queue. Since each map task gets created to process a similar operation and also on similar size of data block they believe to be similar in terms of contribution to temperature rise. Each Task will be assigned to only one task tracker the most so duplicate work being done in entire scene. As shown in Figure 4.3

1. The HeartBeat messages are sent by the Task Tracker i 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 Task Tracker with system commands.

2. The Job Tracker calls Simulator which receives the heartbeat message and extracts the Task Tracker status.

3. In the initial part each Task Tracker has no task so they all will be available to accept a task. So each time a task tracker requests for a task by sending its availability through heartbeat it also registers itself with simulator. If i has just sent heartbeat and asked for a task it will register itself in Simulator.

Figure 4.3: RTAH Design Implementation



4. The Simulator then extracts the message and creates a map structure to keep the heartbeat required fields (CPU_{util} and $Disk_{util}$) with each information being mapped to the Task Trackers address i and their field values.

5. Now on wards each time i task tracker sends heartbeat, simulator updates its map structure to keep information real time.

6. Now simulator has structure with all required fields for simulating the thermal model:

- It uses the CPU usage and Disk usage to calculate the outlet temperature of server i and maintains another map structure for storing the outlet and inlet temperature for Task Tracker.
- Simulator uses model mentioned in equation 4.2 to calculate the outlet Temperature. We uses CPU_{util} as linear model between outlet temperature and $Disk_{util}$ represents either active or idle and it contributes a constant temperature to outlet temperature.
- Now Simulator used outlet temperature to calculate the Inlet temperature for the server itself and also its neighbor task trackers using equation 4.1. T_{sup} is a constant temperature and A is cross interference matrix gives the factor $d_{i,j}$ by which server i is going to contribute to the inlet temperature of its neighbor Task Trackers j. The far the neighbor task tracker is the least the factor in the matrix. The most prominent heat re-circulation affect could be noticed in the local servers in a rack than servers in another rack.

7. 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.

8. The simulator now a boolean method called shouldGo whether scheduler should assign task to the task seeking task tracker. The method here uses the inlet and outlet temperature difference. Lets say task tracker i has requested for task as in equation 4.3.

$$\triangle T^i = {T_{out}}^i - {T_{in}}^i \tag{4.3}$$

9. Now method calculates the temperature rise per map task using equation 4.4. This we can understand by our assumption above that all map tasks are similar so they have very close equal contribution to outlet temperature of server i.

$$\text{Temp rise per map} = \frac{\triangle T}{map_i} \tag{4.4}$$

10. Method now calculates and maintains another structure called possible outlet temperature if job got assigned to that tracker $T_{out}^{possilble}$.

11. Suppose for server i , the method calculates the average of $d_{i,j}$ where j ε [1...n]. Here n is number of Task Trackers in cluster as in equation 4.5.

$$\text{Average Factor for server i} = \frac{\sum_1^n}{n} \tag{4.5}$$

12. From given matrix then scheduler looks for neighbor nodes for which the cross-interference matrix value is greater than average for that server as point out in 4.5. Once known the neighbor servers could be critically affected because of re-circulation effect of heat it then uses $T_{out}^{possilble}$ to calculate the inlet temperature of all its neighbor nodes and stored as $T_{in}^{possilble}$.

13. Now is the moment to decide and make decision of flag the scheduler that whether its good to assign job to server i or not. The criteria being $T_{in}^{possilble}$ for all neighbor nodes of server i and also for i are less than T_{red}. Equation 4.6

$$\text{criteria} = T_{in}^{possilble} < T_{red} \tag{4.6}$$

14. If the criteria in previous step doesn't meet it flags the scheduler don't assign the ready to assign task to this server and wait for next server with heartbeat message ready to be processed.

To Summarize the strategy above is to schedule the tasks to the node which will not bring the inlet temperature for itself and also its neighbor nodes. For making the decision it considers parameters as CPU usage, Disk usage and currently running map tasks on node i.

Technically it considers the Heat re-circulation effect into much in consideration by taking into account the physical layout of data center which is in form of a cross-interference matrix. We also take into consideration of the fact about the physical placement of server with in rack and their proximity to cooling system since that gets affects the Inlet temperature over the course of duration. This strategy works and helps utilizing the thermal management into data center with least affect on performance. Next chapter describes the experiment setup and compiles the results observed and helps strategy put into prospective.

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 10 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. In this section we will be determining the parameters for the model we created in the modeling section for server. Do this we need to prove that all the factors describe in the model will indeed have an effect, and then solve for the constants described in the previously in the modeling section.

5.1 Set up

Using the commodity hardware we setup a Hadoop cluster composed of a Job Tracker, NameNode and 10 DataNodes and Task Trackers.

5.1.1 Hardware

Table 5.1: Server Specifications

Node	Number of cores	RAM	Storage
HP Xeon	4 cores(2.8GHz)	2GB	143GB
Dell	4 cores(2.8GHz)	2GB	143GB

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 12. The data sets are varied as well at sizes of 5GB, 10GB and 20GB 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 chosen to really stress test the cluster by varying CPU and disk utilization by big deal. The Benchmarks used were:

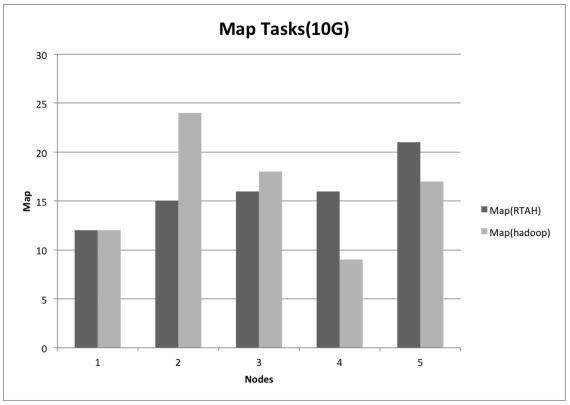
- WordCount
- Distributed Grep
- PI estimation
- Tera Sort

5.2 Results

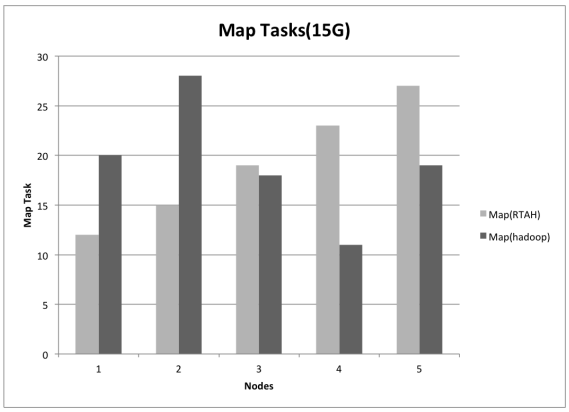
5.2.1 Temperature Reduction

For Evaluation we start with the temperature reduction for nodes which could see imbalanced temperature distribution in default hadoop scheduler which increases their peak

Figure 5.1: Map Task Distribution for WordCount



(a) Number of Map Tasks vs Node(10GB)

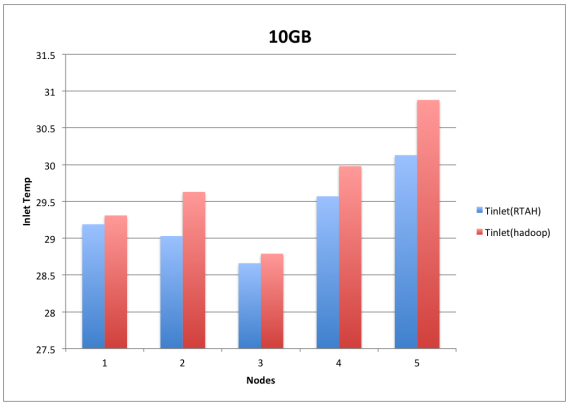


(b) Number of Map Tasks vs Node(15GB)

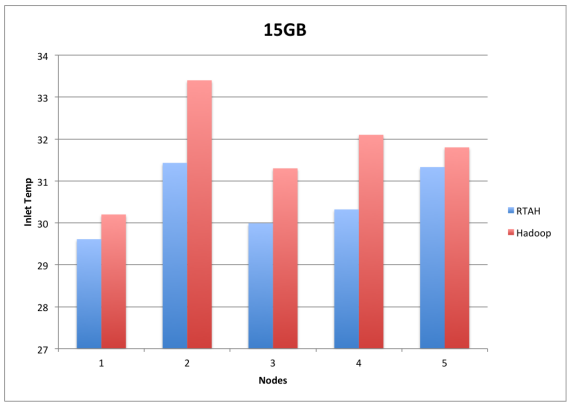
inlet temperature non uniformly, which leads for consistent need of lowering the T_{sup} and power consumption. We stated to test RTAH with original FIFO hadoop scheduler. Figure 5.2(a) shows the number of map task distribution for 5 node cluster processing data size of 10GB and figure 5.2(b) similarly processing the 15GB data size, they both using RTAH and Original Hadoop. We can clearly see that the Original Hadoop doesn't distribute with any sort of knowledge of server location in rack or in data center in general so map task distribution is pretty random but RTAH evidently does consider the location of server in rack if it happens to be close to cooling system or say close to floor it will be cooler in respect to other server;s in the rack so it ends up processing more map tasks than others and with height that number varies it looks like a stair structure. The server far most from CRAC or top server sees the most the heat re-circulation effect so it has its inlet temperature highest at most time of the job processing so it ends up processing the less number of map tasks. We can see here using RTAH we were able to see the peak inlet temperature difference of 1.9 degree Celsius.

In figure 5.3(a) and figure 5.3(b) again evidently shows the prominent reduction in peak inlet temperature of 5 nodes processing two different data size for Word Count application. The inlet temperature reduction helps by avoiding the need of lower cooling temperature which saves the power consumption.

Figure 5.2: Peak Inlet Temperature for WordCount

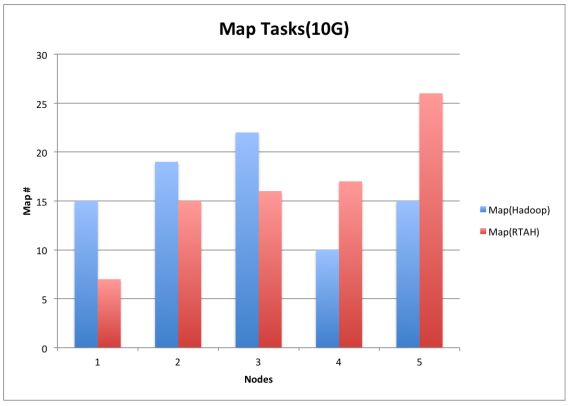


(a) Peak Inlet Temperature vs Node (10GB)

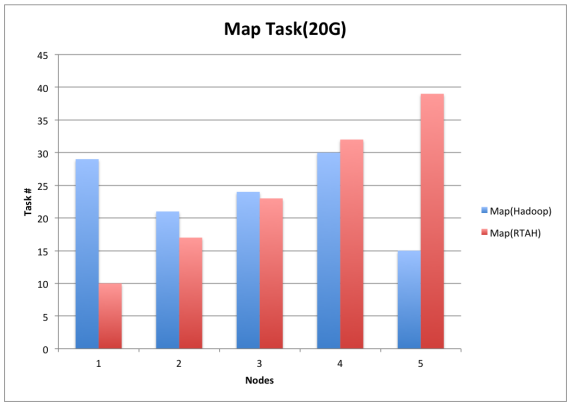


(b) Peak Inlet Temperature vs Node (15GB)

Figure 5.3: Map Task distribution for Grep



(a) Number of Map Tasks vs Node(10GB)

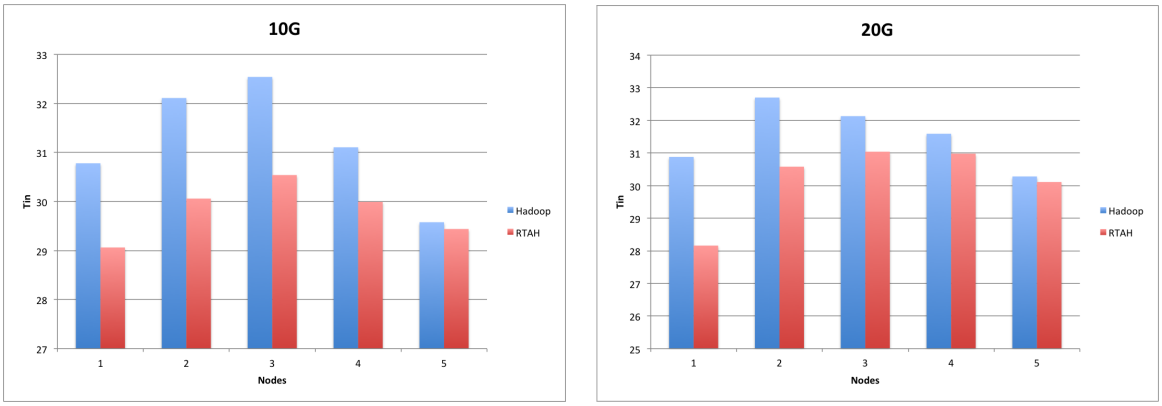


(b) Number of Map Tasks vs Node(20GB)

Similarly for other benchmark application such as grep Figure 5.5(a) and 5.5(b) shows the prominent temperature decline in Peak inlet temperature. We were able to see the temperature difference of 1.5 degree Celsius for grep application. The temperature contribution was primarily by disk usage as grep in more I/O intensive job. From Figure 5.4(a) and Figure 5.4(b) displays that the nodes which were over utilized in FIFO scheduler are literally under utilized in RTAH which could have been the source of temperature imbalance.

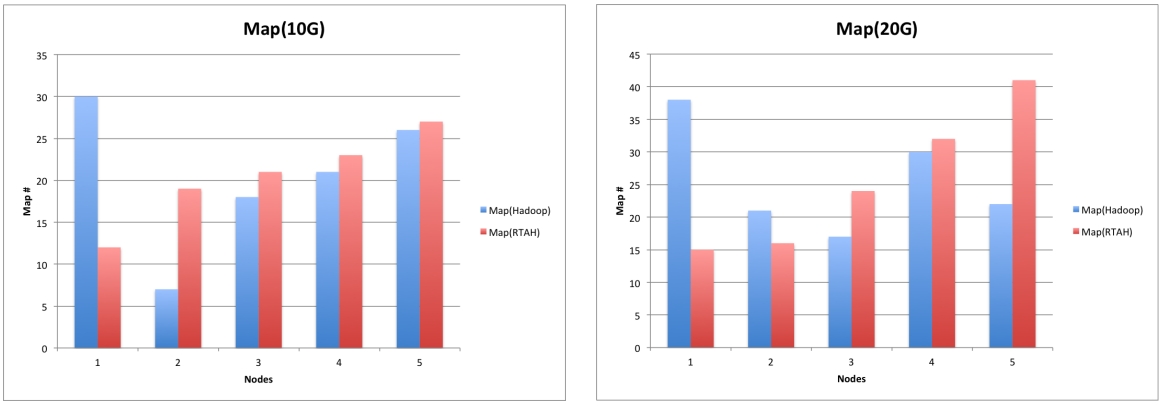
Tera Sort Application [14] which is another industrical standard for benchmarking the Hadoop cluster. Basically, the goal of TeraSort is to sort 1TB of data (or any other amount of data) as fast as possible. It is a benchmark that combines testing the HDFS

Figure 5.4: Peak Inlet Temperature for Grep



(a) Peak Inlet Temperature vs Node (10GB) (b) Peak Inlet Temperature vs Node (20GB)

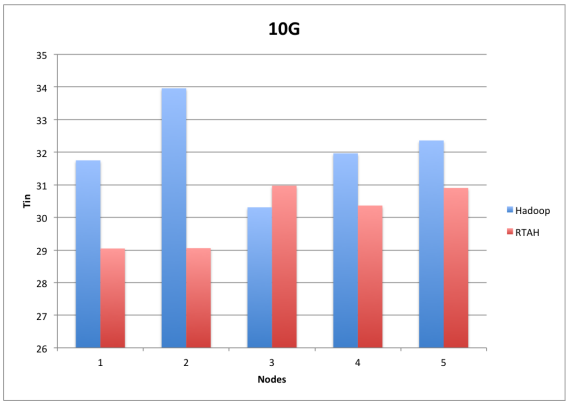
Figure 5.5: Map Task distribution for Tera Sort



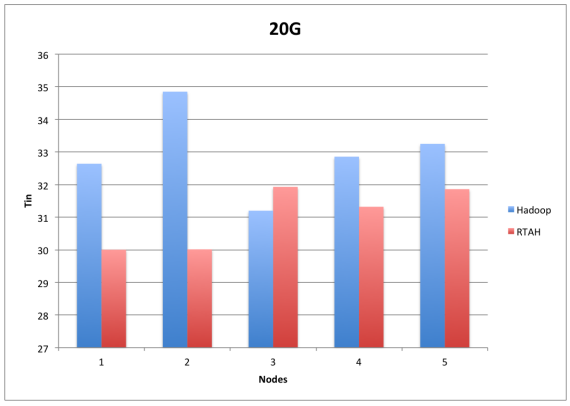
(a) Number of Map Tasks vs Node(10GB) (b) Number of Map Tasks vs Node(20GB)

and MapReduce layers of an Hadoop cluster. As such it is not surprising that the TeraSort benchmark suite is often used in practice, which has the added benefit that it allows us among other things to compare the results of our own cluster with another cluster setting. Tera Sort exhaustively tests the Map Reduce and HDFS interface so its big contributor in faster rise of temperature and then its imbalance. Using RTAH as in Figure 5.7(a) and Figure 5.7(b) shows the prominent decrease in inlet temperature of the cluster's most overused nodes. In a rack like environment they are the ones on the top of rack. RTAH bring down the inlet temperature significantly down by almost 2 degree Celsius.

Figure 5.6: Peak Inlet Temperature for Tera Sort

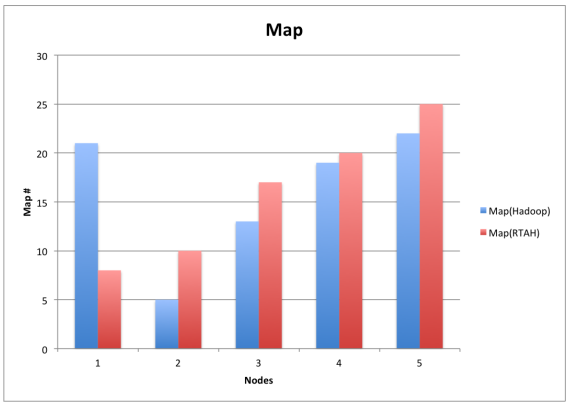


(a) Peak Inlet Temperature vs Node (10GB)

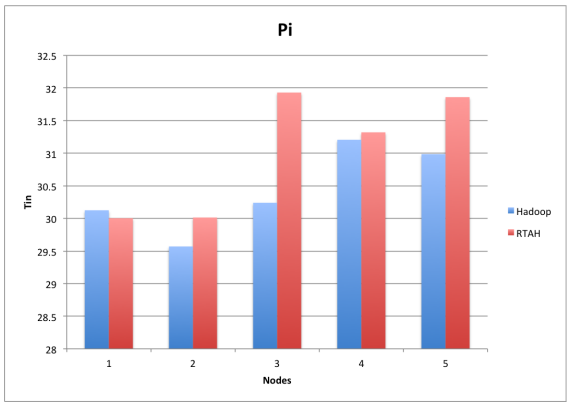


(b) Peak Inlet Temperature vs Node (20GB)

Figure 5.7: Pi Estimation Map Distribution and Peak Inlet Temperature Reduction



(a) Map Task Distribution vs Node



(b) Peak Inlet Temperature vs Node

Figure 5.6(a) and Figure 5.6(b) displays the workload distribution of load in data center to avoid the occurrences or even possibilities of hot sopts.

PI estimation is also another benchmark application benchmark. 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. Figure 5.8(a) and 5.8(b) shows temperature reduction of around 2.2 degree Celcius.

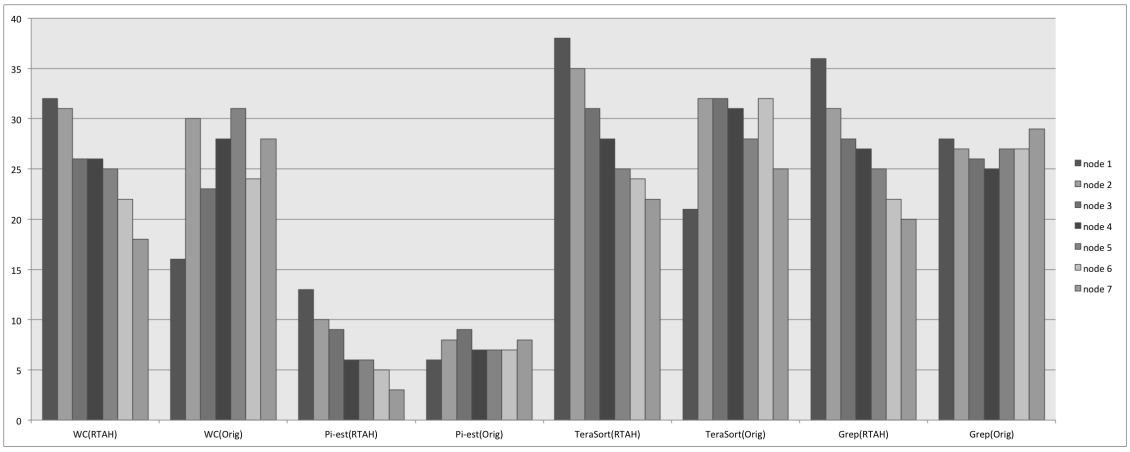
This section of temperature reduction was primarily to display the basic functionality of the scheduler and its minimum goal it met. The results of map task distribution displays the overall performance of the cluster. The nodes overwhelmed with regular FIFO scheduler

tends to be under utilized with RTAH. Purely the reason is that we can see that the nodes which are closer to cooling system should process more task than the ones which are tend to see quick rise in their inlet temperature and which may become the reason for cooling system to bring down the temperature using more power. So this stair like distribution represents the servers in one rack where they do get affected by the neighbor server. The servers on the top are processing less tasks because they are already experiencing temperature rise coming from heat re-circulation effect and they might lead to becoming a hot spot.

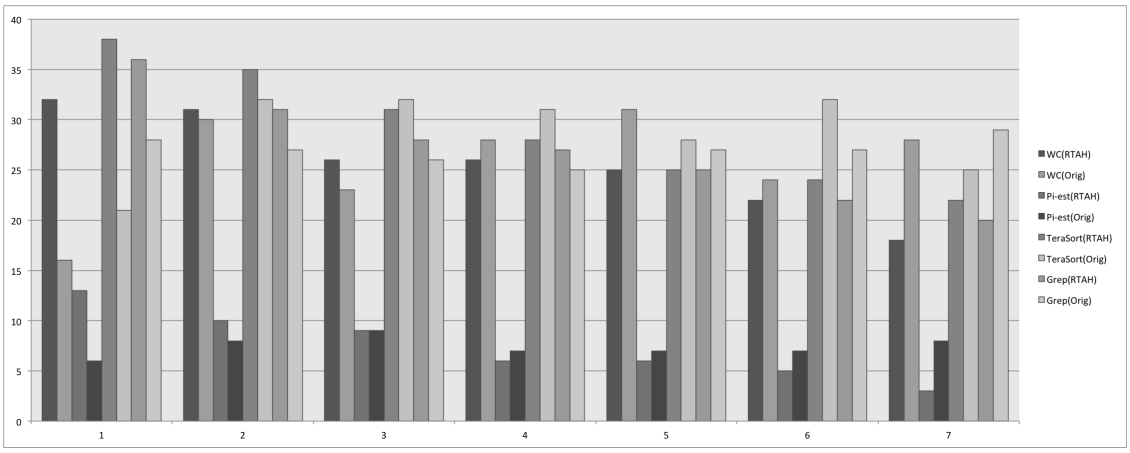
With cluster size 7 and data size of 20 GB as in Figure 5.9(a) shows the level of non uniform distribution of jobs occur with out any thermal management consideration in the hadoop default schedulers, but our thermal model does arrange the tasks in temperature efficient way. In Fig 5.9(b) we can see the distribution of map tasks for each different bench mark we implemented. The difference between the Node 7 and Node 1 which represents the top and bottom servers in the rack respectively and we can the level of contribution height(placement) of server has when it comes to distribution of map tasks. The number of map tasks for pi-estimation is much lesser than others because for other applications the input data size was 20GB while pi-estimation is input data independent it just has computation cost and no data movement at all.

Now here in Figure 5.9 the peak temperature for multiple applications on 7 node cluster. The temperature in original hadoop run and with RTAH peaked more than RTAH. The reason being the server closer to the CRAC is processing more jobs than the one which is far away. So it becomes sort of balance in Peak inlet temperature because the peak inlet temperature for CRAC closer server is high because of its business most of the time and the one far also gets to the similar temperature because of heat re-circulation affect of servers around it. The $Tred$ here is 33.5 degree Celsius.

Figure 5.8: Map Distribution vs Node



(a) Map Task Distribution vs Application



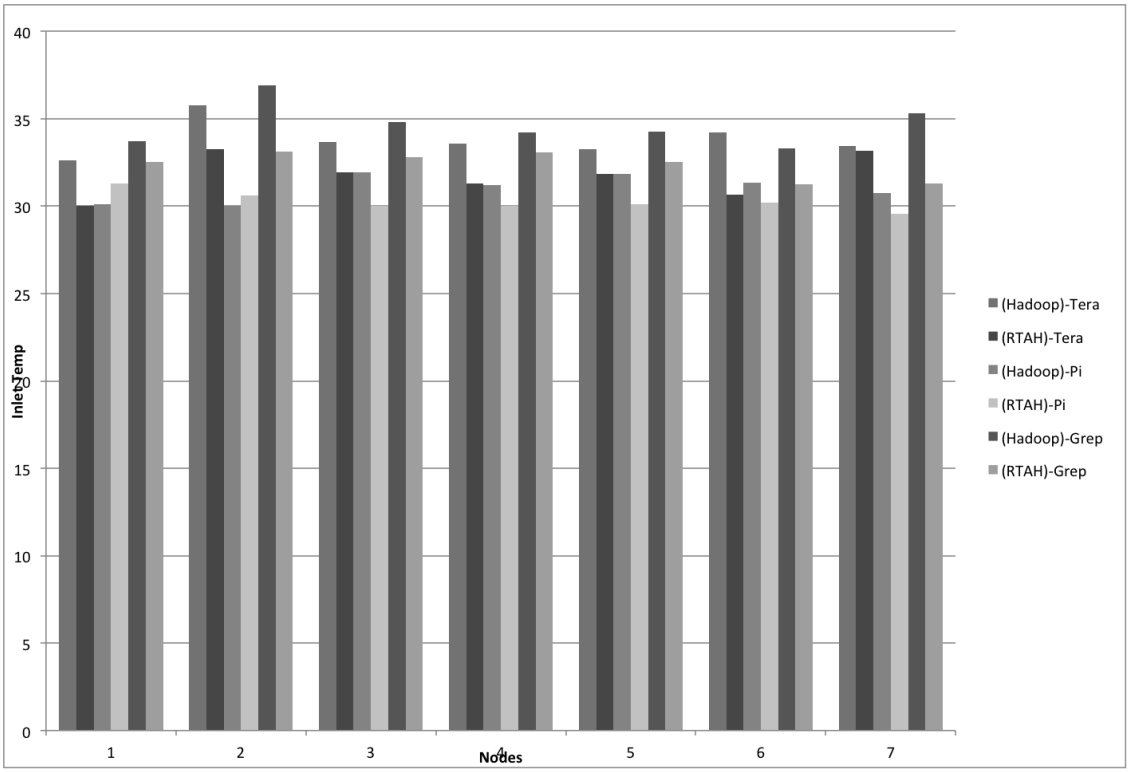
(b) Map-Distribution vs Node

5.2.2 Cooling Cost Estimation

To measure the power consumed and cooling cost. We used the average CPU usage for each node. To calculate the power we used the equation 5.1 where COP($Tsup$) represents the coefficient of performance and P_{AC} shows the cooling cost[23] and T_{sup} here is 20 degree Celsius temperature set up for CRAC temperature. Equation 5.2 shows the used formula to calculate the COP.

$$P_{AC} = \frac{P_c}{COP(T_{sup})} \tag{5.1}$$

Figure 5.9: Peak Inlet vs Nodes



$$COP(T_{sup}) = (0.0068T^2 + 0.0008T + 0.458) \tag{5.2}$$

For calculating the P_c which is computational power consumption used by server itself to for data processing. To calculate we used equation 5.3. As we know that the Total power consumption is sum of computing power and cooling power. Figure 5.10 shows the redcution in difference between T_{out} and T_{in} which is directly related to computing power of server. [23] Now to conclude the cooling cost for all 7 nodes 5.2 which shows the comparable reduction in Cooling power consumption for all four benchmarks used.

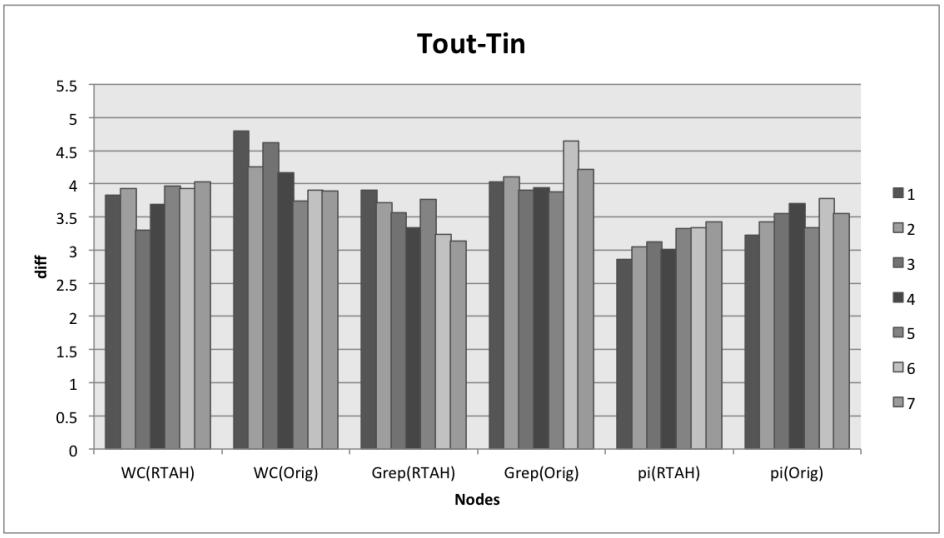
$$P_c = (120 + 50(C_{util}) \tag{5.3}$$

Now for a data center of 50,000 nodes running these benchmarks or some variant of these bench marks will obtain the savings as shown in 5.12 where we can see that the proposed

Table 5.2: Cooling Cost Reduction comparison for Different Benchmarks

	Cooling Cost(in Watt)							
Node	**WC(R)**	**WC(H)**	**TS(R)**	**TS(H)**	**Gr(R)**	**Gr(H)**	**Pi(R)**	**Pi(H)**
1	60.734	48.587	61.294	53.259	59.799	60.734	59.799	56.996
2	59.239	56.062	59.799	60.548	56.996	60.360	58.865	59.239
3	56.062	55.688	58.865	60.734	56.956	59.789	58.304	57.930
4	54.380	56.996	57.557	57.930	56.062	59.052	57.552	58.304
5	53.259	56.249	56.062	57.744	56.060	57.865	56.996	57.557
6	50.455	54.567	53.259	57.557	56.435	57.928	56.622	57.549
7	47.092	57.193	52.324	56.062	56.001	57.183	55.127	59.425
Avg	54.460	55.047	57.023	58.690	56.916	59.132	57.610	58.144

Figure 5.10: Peak Inlet vs Nodes



model will get the cost down significantly. 5.12 shows that we can achieve the cost savings of about $400,000 to $700,000 using the commercial unit price of electricity in states such as Iowa, California and New York.

We also noticed from above data that our model gets power consumption savings from 9% -14%. To make sure not getting the performance affected a lot we also observed the number of times scheduler declines assigning jobs to the task tracer in case when its inlet temperature may exceed red line temperature. We noticed that out of all jobs requested by Task Trackers our Scheduler rejected 3% jobs which is minimal in case of cluster running

Figure 5.11: Cooling Cost Comparison

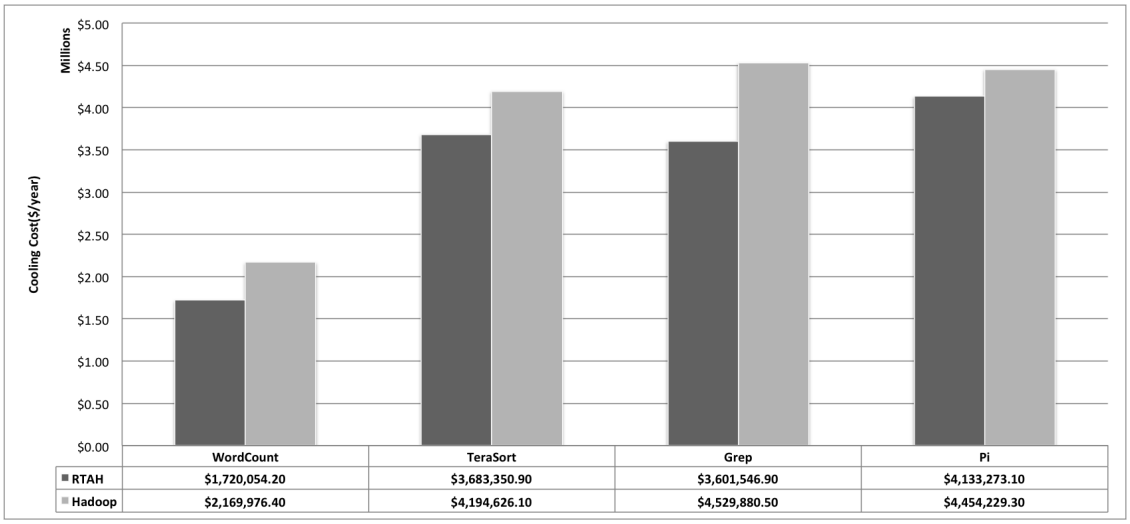


Figure 5.12: Cooling Cost Savings

massive jobs. That many jobs does get black listed in Hadoop's typical processing because of resource non availability out of all jobs. So there is not much performance lag noticed in terms of execution time. If priority of any job is necessary then that can be handled by assigning priorities in application creation which is meant to run on Hadoop and that will further reduce the affect the that small lag. We also noticed the number of jobs rejected

gets less and less further by increasing cluster size which in later end will certainly be more optimal solution. The larger the cluster the more savings we can achieve and also by less affecting the performance.

Chapter 6

Conclusion and Future Work

6.1 conclusions

We have proposed a Thermal Aware Scheduler (TAS) for Hadoop MapReduce framework which extends the First Come First Server (FCFS) scheduler to make it thermally aware. The Thermal Aware Scheduler distributes tasks preferentially to servers based on their heat re circulation affect in their rack they belong to. If it does make any of its neighbor server to exceed threshold temperature it skips assiging job to the task. We then tested our scheduler on four benchmarks WordCount, Terasort, Grep and Pi-estimation which showed that preferential task distribution decreases the peak temperature as well as cooling power. The decrease in the peak temperature was more for WordCount and PI than for TeraSort and grep as former are more CPU intensive.

6.2 Extension

The are several possibilities to extend this work and work on achieving the goal of more thermal and energy efficient data centers.

- Our thermal model approach does take care of Cpu and disk contribution to outlet temperature of a server, we consider the disk to be in a binary state either active or not active. But disk can have also relation with outlet temperature and that relation is certainly not linear so the further extension could be extended to use disk profiling to incorporate the disk contribution in outlet temperature calculation as in our simulator more optimally.

- 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.

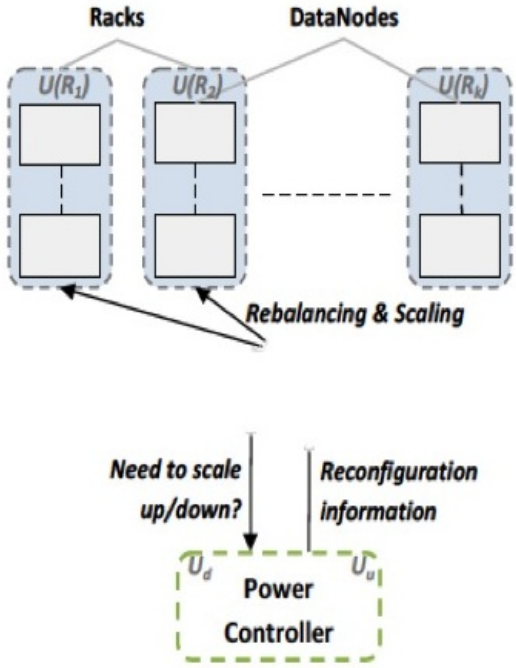
- For making the data center more energy efficient we can come up with a power controller strategy which works on dynamically changing the size of cluster for reducing the overall power consumption of data center. There is one proposed strategy here which could be implemented in Hadoop HDFS data flow to incorporate a power controller.

  - One channel could be implemented between cluster's HDFS and a power controller which together scale up or scale down the number of data nodes dynamically.
  - We know in HDFS the typical replication factor of one block is 3, in which two replica's are local to one rack and third is in some other rack. The strategy has to use the replication factor and its locations to know which nodes to power off and which ones to power on at a particular time.
  - Suppose Name Node recieves a request to create a file F_k of size S_k wit replication factor R_k, it calculates the number of HDFS blocks that will be needed to store it using formula 6.1. where B is block size. The total space needed now is 6.2.

$$\frac{S_k}{B} X R_k \tag{6.1}$$

  - The controller now should check if there is enough space available on HDFS to store the file and turns on more nodes if needed. At all times, There should be some margional space maintained on running data nodes so that write operation doesn't get failed because of less space while then new data nodes being turn on.

Figure 6.1: Power Controller interaction with NameNode



– After the new nodes have been turned on the cluster has to be re-balanced as new nodes are empty at the time. As in Figure **??** from [10]. [9]The name node create HDFS block on newly added data node in rack-aware policy and mean while name space and meta data gets updated in name node records.

– The pure criteria for scaling up or scaling down the cluster is when utilization of one server is below/higher than a threshold utilization.

– At the time of re-balancing the cluster in cases of addition or reduction of a node in cluster, the inter-rack or Intra-rack Transfer of blocks should be critically monitored so that the replication policy of hadoop doesn't get violated.

$$S_k X R_k \tag{6.2}$$

Bibliography

[1] Zahra Abbasi, Georgios Varsamopoulos, and Sandeep KS Gupta. Tacoma: Server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(2):11, 2012.

[2] Abdlmonem H Beitelmal and Chandrakant D Patel. Thermo-fluids provisioning of a high performance high density data center. *Distributed and Parallel Databases*, 21(2-3):227–238, 2007.

[3] Edureka. Apache hadoop hdfs architecture. `http://www.edureka.in/blog/apache-hadoop-hdfs-architecture//`.

[4] eScicence Institute. What is hadoop? `http://escience.washington.edu/get-help-now/what-hadoop`.

[5] Garner. Gartner inc. gartner reveals top predictions for it organizations and users for 2012 and beyond. `http://www.gartner.com/it/page.jsp?id=1862714`.

[6] IDC. Idc press release. `http://www.idc.com/getdoc.jsp?containerId=prUS23177411#.UMYoe-TAfTB,2011`.

[7] 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`.

[8] Xuanfei Jiang. Thermal modeling and data placement for storage systems, 2012.

[9] Jacob Leverich and Christos Kozyrakis. On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[10] Nitesh Maheshwari, Radheshyam Nanduri, and Vasudeva Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Generation Computer Systems*, 28(1):119–127, 2012.

[11] Justin Moore, Jeffrey S Chase, and Parthasarathy Ranganathan. Weatherman: Automated, online and predictive thermal mapping and management for data centers. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 155–164. IEEE, 2006.

[12] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. Making scheduling" cool": Temperature-aware workload placement in data centers. In *USENIX annual technical conference, General Track*, pages 61–75, 2005.

[13] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. Making scheduling" cool": Temperature-aware workload placement in data centers. In *USENIX annual technical conference, General Track*, pages 61–75, 2005.

[14] Michael G. Noll. Hadoop benchmarking. `http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnb`

[15] OReilly. Anatomy of a mapreduce job run with hadoop. `http://answers.oreilly.com/topic/459-anatomy-of-a-mapreduce-job-run-with-hadoop/`.

[16] Chandrakant D Patel, Ratnesh Sharma, Cullen E Bash, and Abdlmonem 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. IEEE, 2002.

[17] Andrea Sansottera and Paolo Cremonesi. Cooling-aware workload placement with performance constraints. *Performance Evaluation*, 68(11):1232–1246, 2011.

[18] Björn Schödwell, Koray Erek, and Rüdiger Zarnekow. Data center green performance measurement: State of the art and open research challenges. 2013.

[19] Ratnesh K Sharma, Cullen E Bash, Chandrakant D Patel, Richard J Friedrich, and Jeffrey S Chase. Balance of power: Dynamic thermal management for internet data centers. *Internet Computing, IEEE*, 9(1):42–49, 2005.

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

[21] 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.

[22] Qinghui Tang, Sandeep KS Gupta, and Georgios Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Cluster Computing, 2007 IEEE International Conference on*, pages 129–138. IEEE, 2007.

[23] Qinghui Tang, Sandeep KS Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *Parallel and Distributed Systems, IEEE Transactions on*, 19(11):1458–1472, 2008.

[24] Rare Mile Technologies. Hadoop demystified. `http://blog.raremile.com/hadoop-demystified/`.