12-2016

Power Management in Heterogeneous MapReduce Cluster

Rojee Sunuwar
University of Nebraska-Lincoln

POWER MANAGEMENT IN HETEROGENEOUS MAPREDUCE CLUSTER

by

Rojee Sunuwar

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Ying Lu

Lincoln, Nebraska

December, 2016

POWER MANAGEMENT IN HETEROGENEOUS MAPREDUCE CLUSTER

Rojee Sunuwar, M.S.

University of Nebraska, 2016

Adviser: Ying Lu

The growing expenses of power in data centers as compared to the operation costs has been a concern for the past several decades. It has been predicted that without an intervention, the energy cost will soon outgrow the infrastructure and operation cost. Therefore, it is of great importance to make data center clusters more energy efficient which is critical for avoiding system overheating and failures. In addition, energy inefficiency causes not only the loss of capital but also environmental pollution. Various Power Management(PM) strategies have been developed over the years to make system more energy efficient and to counteract the sharply rising cost of electricity. However, it is still a challenge to make the system both power efficient and computation efficient due to many underlying system constraints.

In this thesis, we investigate the Power Management technique in heterogeneous MapReduce clusters while also maintaining the required system QoS (Quality of Service). For a cluster that supports MapReduce jobs, it is necessary to develop a PM technique that also considers the data availability. We develop our PM strategy by exploiting the fact that the servers in the system are underutilized most of the time. Hence, we first develop a model of our testbed and study how the server utilization levels affect the power consumption and the system throughput. With the established models, we form and solve the power optimization problem for heterogeneous MapReduce clusters where we control the server utilization levels intelligently to minimize the total power consumption.

We have conducted simulations and shown the power savings achieved using our PM technique. Then we validate some of our simulation results by running experiments in a real

testbed. Our simulation and experimental data have shown that our PM strategy works well for heterogeneous MapReduce clusters which consists of different power efficient and inefficient servers.

DEDICATION

In loving memory of my father who I lost during the course of this degree.

ACKNOWLEDGMENTS

I would like to thank My advisor, Dr. Ying Lu, for her continuous support and guidance throughout the course of this thesis. This thesis would not be possible without her immense knowledge, expertise and supervision. The path to this thesis has been less than smooth and I am very grateful for her unending patience and her invaluable time during the process of its completion. Her kindness and untiring supervision in the face of difficulties always motivated me to work harder. I am forever grateful to her for providing me with an opportunity to work with and learn from her.

I would also like to extend my thanks to my defense committee members Dr. David Swanson and Dr. Lisong Xu for reviewing my thesis. I am very grateful that they took an interest and spent considerable time during the process.

I would like to thank Dr. Swanson further for providing me with the servers and the testbed environment of my own to carry out the experiments. I would like to thank him and everyone in Holland Computing Center who helped me with the installation and setup of the cluster. I am very grateful that they provided help and support with the system without any hesitation whenever something went amiss with the system. I must also thank them for helping me with restarting the system numerous times.

Finally, I would like to take this opportunity to express my gratitude to my family for their support and encouragement throughout my life. They have been a constant source of support and hope for me. I also express my thanks to my older sister for her encouragement and faith in me and more importantly for always motivating me to strive to reach my potential.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the decreasing cost of commodity hardware and increasing amount of data that exists online, data centers are becoming larger than ever and growing at a pace faster than ever. The increasing need for online services, e-commerce and big-data computing has further increased the demand for the data centers to process them. Annually data centers consume enormous amount of energy and spend a lot of money into managing and cooling systems. According to a recent report by US Natural Resources Defense Council [1], US data centers consumed an estimated 91 billion kilowatt-hours of electricity in 2013 alone and they project this figure to grow up to 140 billion kilowatt-hours by 2020. The study also points out that this will cost American businesses 13 billion USD annually for electricity expenses and cause emission of nearly 150 metric tons of carbon pollution. This presents a major challenge for sustainability of the data centers and its future scalability and also emphasizes upon the need to move towards green computing for reducing carbon footprints.

Based on the trends from American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE)[2], it has been estimated that by 2014 infrastructure and energy costs would contribute about 75%, whereas IT would contribute just 25% to the overall cost of operating a data center [3]. This statistic further highlights how the energy cost of

data centers drive the expenses and the need for power efficiency which can significantly decrease the operational cost and contribute to reduced environmental pollution. On the other hand, inefficient power management can also lead to overheating and thus cluster node failures. Overheating causes stress on the cooling system which in turn leads to more power consumption in a data center. Therefore power management of cluster machines is also important for operating cooling systems efficiently, avoiding failures and maintaining system reliability.

The problem of power management in data centers has been a point of interest for industries and is a subject of many research efforts over the past decade. Traditionally, power management of a system involved improving cooling mechanism and packaging technologies while recent ones involve circuit and microarchitectural approaches to reduce thermal stress [4]. Currently many approaches exist to managing power consumption in a cluster and they were suited for different objectives and requirements of the cluster system. Some approaches focus on decreasing the power consumption while guaranteeing some level of Quality of Service (QoS) of the system. Some techniques involve Dynamic Voltage/Frequency Scaling (DVFS) where the frequency and voltage of the system are adjusted to reduce the total power consumption as in [5, 6, 7, 8]. The servers running in lower frequency generate lower heat and hence there will be lower cost for cooling systems too which in turn help to conserve energy in the system. Another popular PM strategy involves designing algorithms for appropriately turning on/off unused or underused servers. But the drawback of such strategies is that the system performance and reliability can suffer adversely. Some approaches like [9, 10] address the problem of data unavailability in such cases. Some existing Power Management solutions seek to reduce the overall or average power consumption of the system, some concern with only reducing the peak power, while others focus on power budgeting [11, 12, 13, 14]. These policies were developed and were applicable to different types of systems. The PM technique that can be adopted for a system depends upon the type of the system and sometimes also

the size of the system. For instance, PM policies developed for homogeneous system may not be applicable to heterogeneous systems [5]. The system load can also determine the applicability and effectiveness of the Power Management policies. The power management strategies differ greatly in clusters that serve only transactional processes like web services versus clusters that serve batch processes like long running programs [4].

In many data centers, the power inefficiency can be attributed to their design. Therefore the PM policy must examine and explore the cause for better power management. Many data centers are designed to handle peak load and over-provisioning of resources is one of the major causes of power inefficiency in data centers [15]. A recent study [16] by Anthesis Group [17] discovered that 30% of servers in data centers are in comatose state where servers are idle for at least 6 months. Therefore over-provisioning is one of the most conspicuous cause for inefficiency in data centers. According to [18], the servers they investigated are rarely idle nor maximally utilized, operating between 10-50% of their maximum utilization levels. As a result, the cluster consumes large amount energy due to the over-provisioning. Thus PM policies can exploit this system under-utilization for better power management and optimize the system for energy consumption through proper resource scheduling. It is for this reason that we were motivated to develop PM mechanism that is based on the system utilization. The fact that the system is underutilized most of the time implies that the capacity of the system is bigger than that needed by the workload. So, we target at such systems to improve their power consumption.

In any under-utilized system it may seem intuitive to turn on/off servers [5, 11] according to the workload demands. But since we use a Hadoop cluster where the files are stored in different nodes, the turning on/off of idle and underutilized servers to reduce power consumption is neither straightforward nor ideal. If all the servers that store a certain data replica used in the job are turned off, this will result in data unavailability. In addition, the turning on/off of servers is not instantaneous and during the state transition servers consume

more power. Another problem with disabling nodes is that it affects cluster performance adversely and may not be applicable for clusters where performance requirements are stringent.

QoS or SLA (Service Level Agreements) in a system is extremely important especially in case of commercial systems offering services to users. According to [19], Amazon found that 100ms of latency cost them 1% in sales while Google found that 500ms in search page generation time reduced traffic by 20%. Therefore it is critical that power management policies take into account the system's QoS. Service providers define and measure their QoS differently. Some of them can also be user defined. For example QoS metrics include mean response time, fault tolerance, availability, throughput, reliability etc. Some systems can also calculate QoS as a combination of different factors.

In this thesis, we focus on power management policy to decrease the overall power consumption of a Heterogeneous MapReduce cluster while maintaining the system's QoS. We define the QoS of our system to be the throughput of the system and seek to guarantee QoS by maintaining the throughput of the system to some desired level. We use MapReduce in our system as it is very popular for data intensive computing. We deploy our cluster in a Linux environment.

Our first step was to model our system appropriately. We target at maintaining the system throughput which depends upon the capacity of the servers. We grouped the servers in our system based on the number of cores in their cpu as the capacity of servers directly depend upon the number of cores. We modeled the power consumption for all the servers in our cluster using an approach similar to those in [11, 20]. Then we also developed the throughput model of our system using their cpu utilization. After the analysis of these models and how the cpu utilization level affected the system, we propose an optimization algorithm to minimize the total power consumption in the system. Our algorithm follows a greedy approach where we seek to utilize the power efficient servers more that the power inefficient ones. Our algorithm works well for a heterogeneous system and its complexity does not

depend upon the number of servers in the system, rather on the number of heterogeneous groups in the system. We first test our algorithm by simulating clusters with workload arrival rates lower than their capacities and calculating the amounts of power savings. Then we generate a MapReduce workload and validate our algorithm and results in a real cluster.

The rest of the thesis is outlined in the following manner. In Chapter 2, we present the background information about the system which includes Hadoop MapReduce and Linux Control Groups, and related works. In Chapter 3, we introduce our models and formally define our power management problem. In Chapter 4, we describe our optimization policy following ordering of servers heuristics and the complete algorithm for achieving our goal of reducing power consumption. Chapter 5 describes our experimental setup and Chapter 6 presents our simulation results and experimental results. We also present analysis and comparison of results obtained by these methods. Finally we conclude in Chapter 7 and provide some insights and details for possible future work in Chapter 8.

Chapter 2

Background and Related Work

In this section we describe the system used in our cluster. We use Hadoop MapReduce in our testbed and use cgroup monitor to enforce maximum cpu utilization bound. We use Linux environment and have installed Scientific Linux 6.8 as the cgroup monitor is available in Linux versions 6 and higher only.

2.1 Hadoop

Hadoop is an open source framework used for a large scale data processing in a distributed environment. It is designed to scale up from single server to thousands of machines with each offering local computation and storage [21]. It has the ability to process large data sets in relatively shorter time and is used as such by wide variety of companies and organizations for both research and production. It is used by companies like Google, Yahoo!, Facebook, Twitter, Amazon for storing, processing and analyzing data.

The Hadoop project includes sub-projects such as Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop YARN and Hadoop MapReduce. Some of them are described in the following sub-sections.

2.1.1 Hadoop MapReduce

Hadoop MapReduce is computational model that splits the large input data-sets into independent blocks and processes them in parallel. The MapReduce processes jobs in two steps, as the name suggests, *map* and *reduce*. First, the map tasks, or mappers, process the blocks of input data according to the map function and output the tuples (key/value pairs). Then these tuples are further processed by reduce tasks, or reducers, which combine them as defined in the reduce function to give the required output. The map tasks are executed before reduce tasks as the output of map tasks are fed as input to reduce tasks. However after some map tasks are finished, the reduce can begin, making the execution of the two stages overlap. Both the input and output of a job are stored in HDFS. The distributed computing model of MapReduce framework allows it to scale data processing quickly over multiple computing nodes. Each node can both process and store data. A MapReduce job can be IO intensive, CPU intensive or Network Intensive depending upon the type of job. The MapReduce framework follows a master-slave architecture. The master is responsible for scheduling, executing and monitoring of tasks in slave nodes.

2.1.2 HDFS

HDFS, short for Hadoop Distributed File System, is a distributed file system used by Hadoop applications. It can also be used as stand-alone general purpose distributed file system [22]. It can store huge amount of data and has the ability to scale quickly. It is highly fault tolerant and is simple in architecture. The replication of data in HDFS provides data redundancy and enables it to overcome high failure rates and ensure data availability. It is designed to span large clusters of commodity servers. HDFS stores data in blocks and the current default block size is 128MB. If the file size is larger than the block size, then HDFS divides and stores the file into multiple blocks [23]. The files smaller than the block size are still stored

in a block and takes the entire block space. If there are a number of small files stored in the HDFS, a lot of space will be wasted. Therefore, HDFS works best with very large files.

2.1.3 Hadoop YARN

Hadoop YARN is a framework for job scheduling and cluster resource management [21]. YARN is short for Yet Another Resource Negotiator [24] and is a new MapReduce implementation designed to addresses the scalability issues with the MapReduce1 (MRv1). In MRv1, jobtracker is responsible for both job scheduling and task monitoring. But in MRv2, these roles are separated using ResourceManager (RM) and ApplicationMaster (AM). RM is responsible for managing resources of the whole cluster and as such manages and coordinates resource allocation across the whole cluster. The resources are expressed in terms of containers which are assigned specific amount of memory and cpu using configuration files. AM is responsible for managing/monitoring the application and its tasks. Each application in YARN has a dedicated AM that runs for the duration of application. It communicates with RM for the resource allocation required for the application. MRv2 also has NodeManager(NM) which runs in each node in the cluster and works with AM to make sure that the resource allocation is followed by all tasks in the node.

Figure 2.1 shows how a MapReduce job is run using YARN. The YARN entities job client, ResourceManager, NodeManager, ApplicationMaster and HDFS are involved and work together to process a job. When the job is submitted to YARN, the job client checks the output specification of the job and computes the input splits required for the job. It then copies the job resources like JAR file, configuration and split information to HDFS. Then the job is submitted. The ResourceManager assigns application ID to the submitted job. Then the ResourceManager invokes scheduler to allocate a container and launches the ApplicationMaster process in that container. In Figure 2.1, MRAppMaster is the

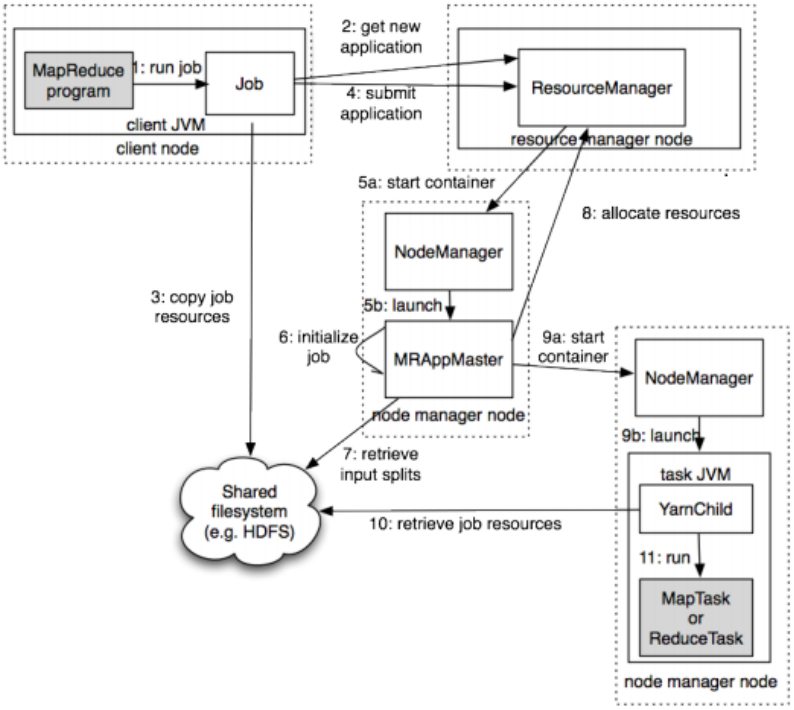

Figure 2.1: Running MapReduce Job using YARN [24]

ApplicationMaster process. The ApplicationMaster is responsible for initializing the job and retrieving the input splits from HDFS. The ApplicationMaster creates the map task and reduce task objects. One map task object is created for each split, so the number of map task objects is the same as the number of splits. For reduce task objects, the number depends upon the mapreduce.job.reduces property in the configuration file. The ApplicationMaster then requests the ResourceManager to allocate containers for all the map and reduce task objects. In ResourceManager, the scheduler assigns containers for them. Then the ApplicationMaster contacts the node manager in each node to launch the containers. Now inside each container, the process YarnChild retrieves the data required for its assigned map or reduce task from HDFS and executes the task. After the job is completed, the ApplicationMaster and the task containers clean up their states and the job information is archived.

2.2 Linux Control Groups

Linux Control Groups, cgroups in short, are kernel features provided in Red Hat Enterprise Linux 6 and higher versions. It is a framework for allocating and managing resources. They can be used for resource allocation such as CPU time, system memory, network bandwidth or combinations of these resources among user defined groups of tasks (processes) running on a system [25]. They give administrators flexibility to allocate the required hardware resources to users, groups and tasks. Cgroups can be used to both assign and deny different amount of resources to different processes. It is a very useful tool for prioritizing tasks when the resources are limited.

Cgroups are organized in a hierarchical structure just like processes. That is, the child cgroups inherit certain attributes from their parents [25]. If the parent processes are assigned some cgroups configuration then the child processes are also constrained under those configurations. There can be more than one cgroups in the system. Therefore cgroups hierarchy consists of one or more unconnected trees of processes. In cgroups, each of the resources are represented by a subsystem. There are 10 subsystems available in Red Hat Enterprise Linux 6. Some of them are cpu - for the CPU access, memory - for system memory allocation and/or limitation, blkio - for input/output access of block devices and ns - for namespace subsystem etc. In this project, only the cpu subsystem is used.

A cgroup hierarchy can have one or more subsystems attached to it. A subsystem can be shared by two or more hierarchies as long as all of those hierarchies have only that subsystem attached to it. That is, a subsystem cannot be shared among two hierarchies if there is another subsystem attached to any one of those hierarchies. A task cannot be a member of two different cgroups from same hierarchy but if the cgroups are located in two different hierarchies, then it is possible. When a task is forked, the child task inherits all the cgroups from its parent task but later on those cgroups can be removed and/or other cgroups can be

assigned to this child task.

The cgroup implementation is provided in Libcgroup package. Libcgroup package consists of different commands for defining and applying cgroups. The cgroups can also be defined in /etc/cgconfig.conf file which remain persistent. Because of this, we use this method to define and mount our cpu subsystem and assign the cpu usage constraints. The details of how different cgroups are implemented can be found in [25]. After defining the cgroups, the tasks should be moved to the cgroups by using the Cgred (Control group rules engine daemon) service. The tasks will continue to use the default resource management of the system until they are specifically moved to the defined cgroup. The Cgred service parameters or rules are set in /etc/cgrules.conf file. Using these rules, we can move all the tasks to required controller i.e. cpu subsystem.

We can use the cpu subsystem to schedule the CPU resources to cgroups. For real-time scheduling tasks, Real-Time scheduler(RT) is used to schedule cpu time while for other tasks Completely Fair Scheduler(CFS) is used. The CFS scheduler proportionately divides the CPU time between tasks on the basis of their priority or shares in their cgroups. It is a relative scheduler and as a result a task can get more than its share of CPU if there are idle CPU cycles. But we are more interested in total CPU utilization rather than prioritizing tasks. CFS also provides a way to define ceiling enforcement or a hard bound on cpu utilization via two parameters: $cpu.cfs\_period\_us$ and $cpu.cfs\_quota\_us$. We can use the first parameter to define the period for CPU reallocation and the second parameter to specify the total amount of time allocated to the tasks in the cgroup. Both time are defined in microseconds. In this way, we can control the maximum cpu resource usage by the tasks. If there are more than 1 CPU in the system, then we need to adjust the amount of time defined in $cpu.cfs\_quota\_us$ as it is evenly distributed among the CPUs.

2.3 Related Work

Extensive research and work has been conducted in the field of reducing power consumption.

Over years many techniques have been developed to reduce power consumption in servers. Some of the techniques utilize scaling of CPU voltage or frequency or both. These are known as Dynamic Voltage and Frequency Scaling (DVFS) or Dynamic Frequency Scaling (DFS). Modern processors like Intel and AMD provide their own implementations. The power consumption is related to clock frequency(f) and cpu voltage(V) as $P = C * V^2 * f$, where C is the capacitance of the digital circuit. These techniques utilize this direct relation and offer to conserve power by reducing server frequency and/or voltage but at the expense of server performance [4, 26, 27]. Depending upon the type of workload, the mean response time and the program speed can suffer negatively. Hence these techniques are generally used only when the workload is not CPU bound [26]. There are significant power management research efforts that use DVFS to save power. Research such as [7, 8] specifically explore DVFS in a MapReduce environment. [8] considers only computation intensive workloads while [7] investigates the effect of scaling cpu frequencies for managing power consumption in Hadoop for different MapReduce jobs.

Another common technique explored is turning on/off of servers. As implemented in [5, 28, 9], this technique calculates the number of servers required at a certain load to satisfy QoS requirement and turns the servers on/off accordingly. This scheme can significantly reduce total power consumption in a cluster as a server can consume more than 50% of the power even at idle state [11, 18]. But turning servers on/off may give rise to another problem - data unavailability, specially for MapReduce workload like ours where input data are stored in the servers and they cannot be randomly turned off.

[9] offers solution to such problem by introducing the concept of "covering subset" mechanism. In this method, the servers are divided into 2 sets, covering and non-covering set.

The covering set contains at least one replica of all data blocks. It allowed the system to turn off at most all servers not in the covering subset. But this solution presents another problem for us. In case of large workload data as in our case, a lot of disk space is wasted storing more replicas of data block. Also this strategy heavily relies upon the replication strategy of the HDFS code which was modified for replica placements. In this method, the state transition from on to off and off to on is not instantaneous. The system consumes more energy when turning on and if the state transition is frequent then it can negate the whole effect of turning servers off to save power. Another challenge of this approach is that it is difficult to design a strategy that won't impact the performance adversely. GreenHDFS [10] proposes another solution by logically separating HDFS into *Hot* and *Cold* zones. They use energy aware placement of data by classifying their data. The highly accessed data was placed into hot zone which contains more servers. The cold zone with very low utilization was then transitioned into inactive power saving modes. The compute power of the Cold zone servers was used only during the periods of peak utilization. In this case too, there is a potential for performance degradation due to the state transitions.

Chen *et al.* [29] present a *Berkeley Energy Efficient MapReduce* (BEEMR) paradigm for an energy efficient MapReduce system targeted for MapReduce with Interactive Analysis workloads. They divide the cluster into disjoint interactive and batch zones. The interactive zone contains a small pool of dedicated servers to serve interactive jobs while the batch zones ran less time-sensitive jobs in a batch fashion. The interactive zone was always fully powered where as the batch zone was put into a low power state to conserve power. [30] presents a MapReduce framework to use green energy. They propose a framework GreenHadoop which predicts the amount of solar energy that will be available in near future and schedules MapReduce jobs accordingly. They use green, solar or wind, energy when they are available and use brown, carbon-intensive, energy only to avoid time violations. This framework can be used whenever green energy is available.

One of the interesting work in Power Management problem is presented by Li *et al.* in [31]. They present a thermal aware solution by exploring the relationship between CPU frequency, temperature and power. It is one of DVFS techniques. They model the power consumption of a system as a function of temperature and frequency. Their strategy is based on the observation that servers consume more power at higher temperature even though they have same frequency and voltage setting and same utilization. They compensate the temperature differences between the servers in the system by decreasing total CPU frequency when temperature increases and increasing it when temperature decreases. We use a similar modeling approach but ours explores the relationship between CPU utilization, throughput and power consumption.

Another related work is by Wang *et al.* [5]. The authors present a PM solution of turning on/off servers by making design decisions on ordered servers list, server activation thresholds and workload distribution. They present different ordering lists for servers to turn on and off. One of the server ordering proposed is Typical Power-based policy where servers are ordered according to their power consumption efficiency under typical workload. While we do not turn on/off our servers, we follow similar approach to order our servers for utilization bound distribution. In our work, the servers are assigned utilization bounds based on their power efficiency at the typical utilization.

The PM strategy presented in [32] also turns off the servers to save overall power consumption but only when there is no work. They propose All-In Strategy(AIS) where the MapReduce job is run on all nodes in the system and the entire system is powered down after the work is completed. But this policy might not work for workloads which arrive in some steady rate that the nodes are always lowly utilized. If the workload arrives intermittently, they put the jobs in queue and submit them in one batch after a certain threshold is reached. This can cause significant delay for early arriving jobs which may not be acceptable. Also the benefits of this policy might become overshadowed by the cost of state transitions.

Yigitbasi *et al.* [33] present a solution for energy efficiency in MapReduce cluster by proposing scheduling heuristics. They try to schedule the jobs in most energy efficient node which has free slots. They consider the heterogeneity of the MapReduce cluster similar to ours. They do not explore the energy efficiency of the cluster when the workload are arriving at different rates. In our work we incorporate the workload arrival times as it applies to the power management policy. We include all the nodes to run the MapReduce workload as in [32] which are ordered in a fashion similar to [5].

Chapter 3

Problem Description

3.1 Power model

The power consumption of a server can be broken down into the power consumed by its components cpu, memory, hard drives, network cards etc. The amount of power consumed by each component at peak depends upon the system type (small vs large), architecture(single-core vs multi-core) and manufacturer. Figure 3.1 shows the power consumption for different components in a Quadcore Intel Xeon server. In this case, the most power is consumed by processor and memory.

In this research we focus on the power consumed due to CPU utilization and using it to

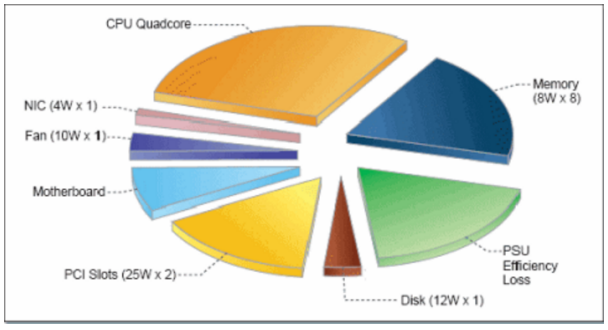

Figure 3.1: Power Consumption of components in a server [20]

minimize the total power consumption.

The power consumption, P, is linearly related to its cpu utilization u and is approximated as given in [11, 20].

$$P = (P_{max} - P_{min}) * u + P_{min} \tag{3.1}$$

P_{min} also known as static power, is the power consumed when $u = 0\%$. It is the power consumed when the server is not in use. P_{max} is the power consumed when the server is in full use i.e. when $u = 100\%$. In equation 3.1, the value of u is expressed in percentage and ranges from 0% to 100%. This method can accurately approximate power consumption with error margin of $\pm 5\%$ [20].

In general, the power consumption, P_i, of a server i can be expressed as a function of its cpu utilization u_i as,

$$P_i = \alpha_i u_i + \beta_i \tag{3.2}$$

where,

$\alpha_i = P_{i_{max}} - P_{i_{min}}$

$\beta_i = P_{i_{min}}$

$u_i =$ CPU utilization

We refer to α_i and β_i as power coefficients and their values depend upon each individual server. As the equation shows, the server utilization and power are linearly related and therefore consumes a higher power at a higher utilization. Apart from utilization, the amount of power consumed is also dictated by their power coefficients. The servers with lower values for power coefficients are considered more power efficient. As a result, to minimize the power consumption of a cluster, our approach is to achieve higher utilization on power efficient servers and lower utilization on less efficient ones. But it is also important to note that a server consumes more than half of its full power even at its idle state. We want to investigate whether or not this is an effective approach to save power.

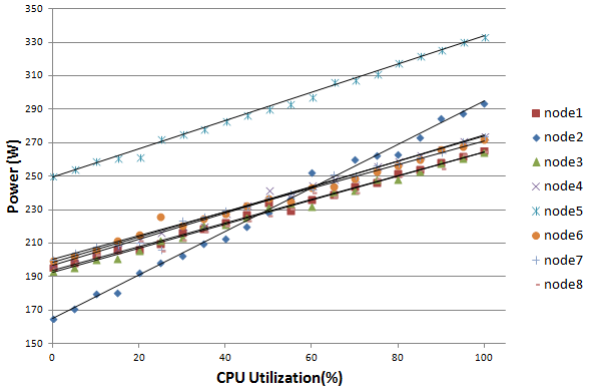

Figure 3.2: Power Consumption Model of 8 Servers in Our Testbed

Figure 3.2 shows the power equation models of 8 different servers used in our testbed. Each server is measured for power consumption at different CPU utilization level and the points are plotted in the graph. The points were then approximated for trend line which gave us a linear model. The plots for different servers can be differentiated with different plot point shapes. Our measurements show that the power approximation calculated on average in this form for all servers is accurate within an error margin of just $\pm 1\%$.

3.2 Throughput Model

The throughput of a server is directly affected by its CPU-utilization. Therefore, similar to the power model, system throughput can be modeled using its CPU utilization. The throughput(τ_i) of server i, is related to its utilization u_i as $\tau_i = \gamma_i u_i$, where γ_i is the throughput coefficient. Our experiments have shown that this model fits our observed data well. The value of γ may vary for different servers and the server with a higher value is considered more efficient. Such a server can give a higher throughput for a given CPU utilization due to the linear relationship. The server with a higher number of CPU cores can process workload at a higher rate and thus has a higher value of γ and is more throughput efficient. For a system consisting of many servers, its throughput can be assumed to be equivalent to the summation

of τ_i of each server i in the system. That is, the total throughput of the system τ can be calculated as:

$$\tau = \sum_{i=1}^{N} \tau_i = \sum_{i=1}^{N} \gamma_i u_i \tag{3.3}$$

The maximum total throughput τ_{max} is achieved when all servers in the cluster are running in their full capacities i.e. $u_i = 100\%$ for all servers i. But the level of utilization also depends upon the incoming workload λ in the system. If the arrival rate of the current workload is less than the maximum throughput capacity then the system should be able to process them at a speed same as the arrival rate. The throughput of the system in such a case will be equivalent to the arrival rate of the workload. For a system with a higher arrival rate than the maximum capacity of the server, the system will be fully-utilized and its throughput will be at its maximum level as jobs are placed in the queue waiting to be processed later.

3.2.1 Throughput Model for heterogeneous system

We define a homogeneous system to consist of servers with similar throughput capability and a heterogeneous system to consist of dissimilar ones. In a homogeneous system, all the servers will process data at the same rate because of their same or similar throughput capability. In heterogeneous system, however, the rate at which different capacity server processes data can vary significantly. The servers which can produce higher throughput will process data at higher rates. In such a case, if we direct more workload towards higher capacity servers then as a result the throughput of the system will also become higher. For a high workload, it may not offer any discernible difference but for a light workload, the throughput can be maximized.

We plan to operate different servers at different utilization levels. We can control the CPU utilization by using cgroups. The cgroup containers can restrict the processes in the containers to use up to a given quota of cpu resource and hence provide the cpu utilization

ceiling cut off. Keeping this in mind, we group servers in the system by their throughput capabilities. Different servers can have different values for throughput coefficient γ and no two servers may have the exact same value. So we group servers with similar throughput capability together in the same group instead of looking for the exact same value. From our experiments we have observed that the value of γ is largely affected by the number of cores in a server, with a higher core count leading to a higher throughput as there are more cores to work on the tasks.

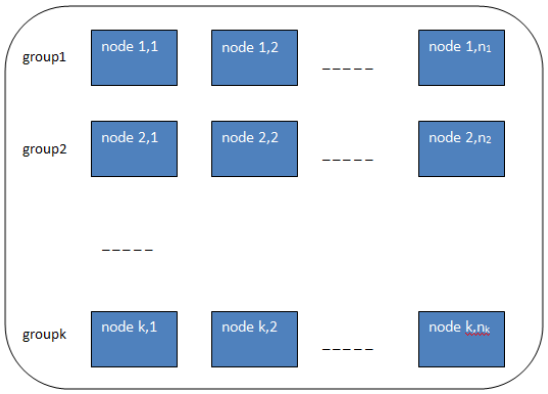

Figure 3.3: Grouping of servers into k different groups

Using this methodology, we group our servers into different groups. Figure 3.3 shows the grouping of servers into different groups. We also extend the throughput model from section 3.2 to be the sum of throughput of different groups. Within each group, the throughput will be the sum of the total throughput contribution of all the servers in the group. To make the system power efficient, we want to vary the utilization level for each server in the group. But in terms of throughput, we represent this group of server as if it were a single server with utilization level equivalent to the sum of all the utilization of the servers within the group.

Let us assume that there are N servers in a system and they are grouped into k groups according to their throughput capabilities. Let $\hat{u}[i]$ be the sum of CPU utilization of all the

servers in group i, then we model the throughput of this system to be:

$$\tau = \sum_{i=1}^{k} \hat{\tau}[i] = \sum_{i=1}^{k} a_i \hat{u}[i] \tag{3.4}$$

where a_i is the representative throughput coefficient of group i. It is important to note that each group may consist of different number of servers.

3.3 Problem Definition

As with many Power Management problem, our objective is to solve a PM problem following the QoS requirement of the system. We define the QoS requirement on the throughput of the system. The purpose is to minimize the total power consumption of the cluster without significantly affecting the system throughput. We assume that the cluster has a light workload, that is, the incoming workload does not overwhelm the servers and is always less than the maximum operating capacity of the cluster. This is often the case for computing clusters used in both academia and industry. We also assume the system to be heterogeneous. The power and throughput are modeled as described in above sections and are directly related to the CPU utilization. The main purpose of this thesis is to find the optimal maximum utilization configuration for individual servers in a heterogeneous cluster so as to minimize the total power consumption P, while maintaining the throughput τ.

In our problem, we assume that the cluster is not overloaded i.e. it is always the case that $\lambda \leq 95\%\ \tau_{max}$, where τ_{max} is the maximum attainable throughput of the system. The reason for such an assumption is that for higher workload, the cluster may not be able to process the jobs efficiently if cpu utilizations are kept at lower values. The system throughput will suffer too much.

For a cluster of N servers, let us formally define our objective as follows

minimize

$$P = \sum_{i=1}^{N} (\alpha_i u_i + \beta_i)$$

subject to:

$$\tau = min\{95\%\tau_{max},\ \lambda\}$$

$$30\% \leq u_i \leq 100\%, \qquad i = 1, \ldots, n. \tag{3.5}$$

where the cpu utilization u_i is bounded between 30% to 100%. If a server's utilization is set to 0%, that server will not be used and will be excluded from the job computation. As mentioned in previous sections, we are using Hadoop MapReduce jobs as our workload that read input data from disks. We do not plan on turning off any of our servers as there should be at least n replicas to turn off $n-1$ servers to ensure data availability[9]. Therefore to avoid servers from consuming static power without contributing to the overall computation, we fix the value of minimum utilization u_{min} to be 30%. According to our experiments and collected data, we found that this value is appropriate and do not cause network congestion while ensuring that all the servers are used in all cases. The highest possible utilization bound u_{max} of server is at 100%.

In the throughput model, the servers were grouped into k groups according to their throughput coefficient. Similarly, we can represent total power to be the sum of the power consumed by these groups, $P = P_1 + P_2 + \ldots + P_k$. These groups correspond to the groups of servers in the throughput model. If there are n_1 servers in group1 and n_2 servers in group2 and so on then the total power consumption can be given by, $P(t) = \sum_{i=1}^{n_1}(\alpha_{1,i} * h_{1,i}(\hat{u}[1]) + \beta_{1,i}) + \sum_{i=1}^{n_2}(\alpha_{2,i} * h_{2,i}(\hat{u}[2]) + \beta_{2,i}) + \ldots + \sum_{i=1}^{n_k}(\alpha_{k,i} * h_{k,i}(\hat{u}[k] + \beta_{k,i})$ where $h_{1,i}(\hat{u}[1])$, $h_{2,i}(\hat{u}[2])$ and $h_{k,i}(\hat{u}[k])$ are utilization distribution functions in group1, group2 and groupk respectively.

These distribution functions are described in detail in section 4.3. For light system where the arrival rate λ is always less than 95% of the capacity of the system, it is sufficient to satisfy $\tau \approx \lambda$.

Now our objective function becomes,

$$\begin{aligned} &\text{minimize} \\ &P(t) = \sum_{j=1}^{k} \sum_{i=1}^{n_j} (\alpha_{j,i} * h_{j,i}(\hat{u}[j]) + \beta_{j,i}) \\ &\text{subject to:} \\ &\begin{cases} \lambda = \sum_{j=1}^{k} a_j \hat{u}[j] \\ n_j * 30\% \leq \hat{u}[j] \leq n_j * 100\%, \qquad j = 1, 2, \ldots, k \\ n_1 + n_2 + \ldots + n_k = N \end{cases} \end{aligned} \tag{3.6}$$

where the throughput constraint is updated with the throughput model equation. The utilization constraints are also expressed in terms of group utilization. Since the utilization u_i of each server i is bounded as $30\% \leq u_i \leq 100\%$, the utilization of a group j with n_j servers will also be bounded as $n_j * 30\% \leq \hat{u}[j] \leq n_j * 100\%$. Lastly the final constraint defines that the sum of the number of servers in all groups should be equal to the total number of servers in the system.

Chapter 4

Optimization

This section describes the power optimization of the system. Both the power consumption and the throughput of the cluster are dependent upon utilization level of servers in the cluster. Both of them are maximized when the cluster is fully utilized. To minimize the total power consumption, we want to control the maximum utilization of different servers. But, doing so will also decrease the throughput of the cluster. Therefore, we need to consider the trade off between power consumption and throughput of the system. Our strategy is to minimize the utilization of the power inefficient nodes while keeping the throughput of the system to a certain level.

The objective function as given by equation 3.6 depends upon the utilization constraint of each server. Therefore the optimization tools such as MathProg or Simplex method cannot be used to solve this problem as it will be exponential time with the number of servers. Therefore we develop a heuristic algorithm to find the optimal assignment of utilization bound for each server while also achieving certain throughput for the system. As servers are grouped into different groups, the optimization problem can be divided into two steps. The first step is to find the total utilization bound for each group and the second step is to find a

way to distribute the group utilization bound within the group. But before describing the optimization algorithm, let us first look at the ordering policy of servers.

4.1 Ordering Servers

The servers in different groups should be ordered in some way for distributing the utilization bound in the group such that the overall power consumption is minimized. We want to choose a policy such that the power efficient servers are assigned higher utilization bound than the power inefficient servers in the same group.

Different servers with similar throughput capacity may have different power coefficients. We know that power consumption of each server with respect to their cpu utilization is given by $P_i = \alpha_i * u_i + \beta_i$. It is clear that smaller value of α_i and β_i yields lower power consumption and the servers can be listed in the increasing order of these values. But for any two servers i and j, if $\alpha_i < \alpha_j$ and $\beta_i > \beta_j$, then their power consumption ordering is not fixed. In such case we need to order them by some other criterion. We pick their power consumption at a typical utilization for ordering the servers within a group, similar to the ordering policy adopted in [5].

The lowest utilization is 30% and the highest is 100%. So we assume the mid-point i.e. 65% to be our typical cpu utilization. Then we order our nodes based on their power consumption at this utilization level (u_t). This ordering is entirely static and depends upon both α_i and β_i. It is fixed beforehand and is carried out separately for each group.

4.2 Finding Group Utilization

Let us assume that the servers are divided into k groups according to their throughput capacity. Each group can have variable number of servers. Within each group servers are ordered by the non-decreasing order of their total power consumption at the typical utilization level. The initial state of the system is as given in algorithm 1.

Algorithm 1 Group Utilization Bound Determination

Initialization:
$\lambda = workload\ arrival\ rate$
$k = number\ of\ groups$
$n_1, n_2, \ldots, n_k : number\ of\ servers\ in\ each\ group$
$u_{min} : minimum\ utilization\ of\ an\ individual\ server = 30\%$
$u_{max} : maximum\ utilization\ of\ an\ individual\ server = 100\%$
$temp[i] : temporary\ utilization\ solution\ for\ group\ i$
$P_{min} : initialized\ to\ be\ = \sum_{i=1}^{N} (\alpha_i * 100\% + \beta_i)$
Constraint:
$\lambda = \sum_{i=1}^{k} a_i \hat{u}[i]$
Output:
$\hat{u}[k] : group\ utilization\ bound\ for\ each\ group$
for each group i **do**
    $u_{low}[i] = n_i * u_{min}$
    $u_{high}[i] = n_i * u_{max}$
    $temp[i] = 0$
end for
$FindGroupUtilizationBound(k)$

In linear optimization, the optimal value lies at one of the endpoints. Our optimization algorithm is based on this property of the linear optimization. We set each individual server's utilization bound to a value from 30% to 100%. The value of total utilization bound in each group is also bounded in a range as given by the constraint equations. This default range is too large. Our algorithm tries to decrease this range by using the throughput constraint. For any given throughput(or the arrival rate) λ, the new range of one group utilization can be calculated using the default ranges of the other group utilization. Then, the utilization

of other groups can be calculated using this new range and other default ranges. In every step, one default range is replaced by the calculated range until new ranges for all groups are calculated. For example, at the beginning the default utilization range for the last group k, $\hat{u}[k]$ is:

$n_k * 30\% \ \leq \ \hat{u}[k] \ \leq \ n_k * 100\%$

Using λ, the new range for the group utilization $\hat{u}[k]$ becomes:

$(\lambda - \sum_{j=1}^{k-1} a_j * u_{high}[j])/a_k \ \leq \ \hat{u}[k] \ \leq \ (\lambda - \sum_{j=1}^{k-1} a_j * u_{low}[j])/a_k$

Simplifying it we get,

$max\{n_k * 30\% \ , \ (\lambda - \sum_{j=1}^{k-1} a_j * u_{high}[j])/a_k\} \leq \hat{u}[k] \leq min\{n_k * 100\% \ , \ (\lambda - \sum_{j=1}^{k-1} a_j * u_{low}[j])/a_k\}$

where $u_{low}[j]$ and $u_{high}[j]$ are default lower and higher endpoints for the j^{th} group respectively.

As the optimal value always lies at one of the endpoints, we choose one of the endpoints as the possible value for $\hat{u}[k]$, represented as $temp[k]$, and calculate the new ranges for other groups based on this value. From each endpoint, we will get a different solution. For the other groups we use the new endpoint values if they have been computed otherwise we use the default ones. Among these solutions, the one that leads to the lowest total power consumption is picked as the optimal solution for the group utilization bound.

The function FindGroupUtilizationBound calculates the total group utilization bound for all groups recursively as described above. The function $h_{j,l}(temp[j])$, which is a linear function, returns the utilization bound u_l for the l^{th} server in the j^{th} group as given by algorithm 3.

Algorithm 2 FindGroupUtilizationBound(i)

1: **if** $i > 1$ **then** find the new endpoints as:
2: $u_{low}[i] = max\{n_i * u_{min},\ (\lambda - \sum_{j=1}^{i-1}(a_j * u_{high}[j]) - \sum_{j=i+1}^{k}(a_j * temp[j]))/a_i\}$
3: $u_{high}[i] = min\{n_i * u_{max},\ (\lambda - \sum_{j=1}^{i-1}(a_j * u_{low}[j]) - \sum_{j=i+1}^{k}(a_j * temp[j]))/a_i\}$
4: **for each** $b \in \{u_{low}[i], u_{high}[i]\}$ **do**
5: $temp[i] = b$
6: $FindGroupUtilization(i-1)$
7: **end for**
8: **else**
9: $temp[i] = (\lambda - \sum_{j=2}^{k}(a_j * temp[j]))/a_i$
10: **if** $temp[i] < u_{low}[i]$ **or** $temp[i] > u_{high}[i]$ **then**
11: not a feasible solution.
12: **else**
13: $P = \sum_{j=1}^{k}\sum_{l=1}^{n_j}(\alpha_{jl} * h_{jl}(temp[j]) + \beta_{jl})$
14: **if** $P < P_{min}$ **then**
15: $P_{min} = P$
16: **for each** group j **do**
17: $\hat{u}[j] = temp[j]$
18: **end for**
19: **end if**
20: **end if**
21: **end if**

4.3 Utilization distribution in a group

After getting the utilization value for each group, they need to be distributed to the individual servers in each group. The final output should be the utilization value u_i for each server i. This algorithm describes the distribution of group utilization bound $\hat{u}$ to each server in the group. This distribution algorithm is based on the fact that power efficient servers should be assigned higher utilization bounds whereas lower efficient ones should be assigned lower bounds. Our target is to use power efficient nodes as much as possible. Therefore our utilization bound distribution algorithm follows a greedy heuristics where the utilization bound is distributed in order of the power efficiency. That is, the power efficient servers are given as high utilization bound as possible while also ensuring at least the minimum utilization bound of 30% to the power inefficient servers.

In such case, some servers (say m) are assigned 100% while all other servers are assigned at least 30%. The next server $m+1$ will be assigned the remaining utilization.
The power consumption of a group with n servers is given by:
$P = \sum_{i=1}^{n}(\alpha_i * h_i(\hat{u}) + \beta_i)$
Then following the utilization bound distribution, it can be expanded as:
$P = \sum_{i=1}^{m}(\alpha_i * 100\% + \beta_i) + (\alpha_{m+1} * u_{m+1} + \beta_{m+1}) + \sum_{i=m+2}^{n}(\alpha_i * 30\% + \beta_i)$

where we ensure that $u_{m+1} = \hat{u} - m * 100\% - (n - m - 1) * 30\%$ and $u_{m+1} \geq 30\%$.

If the total utilization bound to be distributed is $\hat{u}$, then we need to find the server m, such that all servers that are as efficient are assigned maximum utilization bound i.e. 100% and all servers that are less efficient than server m are assigned minimum utilization bound i.e. 30%. Then the server $m+1$ will be assigned the utilization bound which can be found out by calculating the remaining value.

Algorithm 3 Utilization Bound Distribution in a Group

Initialize:
    $S = \bigcup_i s_i, \quad \{i = 1, 2, \ldots, n\}$ includes all servers in a group, ordered in non-increasing order of power efficiency
    $\hat{u}$ = total utilization to be distributed
    $totalMin = n * u_{min}$
    $totalMax = n * u_{max}$
if $\hat{u} > totalMax$ **then**
    Error value for $\hat{u}$
else if $\hat{u} < totalMin$ **then**
    $u_i = 30\%, \quad i = 1, 2, \ldots, n$
else
    $m = \lfloor (\hat{u} - totalMin)/(u_{max} - u_{min}) \rfloor$
    for each node $s_i \in S$ **do**
        if $i \leq m$ **then**
            $u_i = u_{max}$
        else if $i > m + 1$ **then**
            $u_i = u_{min}$
        else
            $u_i = \hat{u} - m * u_{max} - (n - m - 1) * u_{min}$
        end if
    end for
end if

Chapter 5

Experimental Setup

Our testbed consists of 9 server nodes with 1 masternode and 8 slave nodes placed in a single rack. The Apache Hadoop Yarn [34] version 2.5.2 is installed in our cluster with operating system environment Scientific Linux version 6.8. The nodes are heterogeneous considering cpu core configuration.

The nodes are grouped into 2 groups according to their cpu-core configurations. Group1 consists of a single server node1 with an 8-core cpu while group2 consists of 7 nodes each with a 4-core cpu. The group1 server has Quad-Core 2.2 GHz AMD Opteron(tm) Processor 2354 (64 bit) with 8GB RAM and all servers in group2 has Dual-Core 2.8 GHz AMD Opteron(tm) Processor 2220 (64 bit) with 8GB RAM. They are all connected with 1Gbps Ethernet. The head node also has the same configuration as the servers in group2. Head node is used for submitting workloads but is not used to process MapReduce jobs. It is also used as a logging node to log the cpu utilization and power consumption of all servers in the testbed. In this chapter, we present modeling and setup of our testbed for conducting experiments and simulations.

5.1 Power Modeling

We collected the power consumption data for each server and modeled their power equation with respect to the cpu utilization accordingly. We used Server Tech power distribution unit (PDU) to measure the power from the servers. We observed that the power consumption had a linear relationship with the utilization. The power was measured using PDU by taking power usage samples every 5 seconds for some duration and taking average of those power measurements. These measured power data were plotted against the corresponding utilization data as shown in Figure 3.2. The power model equations approximated using our measured data are given in Table 5.1. From the table, we can observe that for node1 which has 8-core cpu, the power equation is a little different from the others.

Table 5.1: Power Equation Models of Testbed Nodes

node	α	β	Equation
1	129.56	165.26	P = 129.56*u + 165.26
2	70.08	194.93	P = 70.08*u + 194.93
3	71.58	192.84	P = 71.58*u + 192.84
4	77.48	196.99	P = 77.48*u + 196.99
5	84.09	249.17	P = 84.09*u + 249.17
6	70.47	200.6	P = 70.47*u + 200.60
7	75.39	198.79	P = 75.39*u + 198.79
8	70.3	194.05	P = 70.30*u + 194.05

We also validated the power data calculated from the above power equation against the measured data. We found that the error is approximately $\pm 2\%$ in every case and less than $\pm \%1$ on average. The following tables show our measurement versus the approximated data and the error percentage for every calculation.

Table 5.2: node1 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	165.56	165.26	0.18
5	171.39	171.74	0.20
10	180.65	178.22	1.35
15	181.36	184.69	1.84
20	193.10	191.17	1.00
25	199.09	197.65	0.72
30	203.31	204.13	0.40
35	210.12	210.61	0.23
40	213.33	217.08	1.76
45	220.82	223.56	1.24
50	228.92	230.04	0.49
55	237.46	236.52	0.40
60	252.65	242.99	3.82
65	245.00	249.47	1.83
70	260.66	255.95	1.81
75	262.97	262.43	0.21
80	263.43	268.91	2.08
85	273.82	275.39	0.57
90	285.01	281.86	1.10
95	288.29	288.34	0.02
100	293.89	294.82	0.32

Table 5.3: node2 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	193.72	195.68	1.01
5	197.18	198.79	0.82
10	201.45	203.31	0.92
15	205.77	206.56	0.38
20	207.11	206.60	0.25
25	209.26	210.04	0.37
30	216.26	216.67	0.19
35	218.52	218.86	0.15
40	221.91	222.62	0.32
45	225.69	227.18	0.66
50	232.05	234.58	1.09
55	232.28	229.95	1.00
60	234.42	236.36	0.83
65	237.74	239.30	0.66
70	242.02	244.30	0.94
75	246.31	246.58	0.11
80	253.95	251.68	0.89
85	255.26	254.10	0.45
90	260.33	258.59	0.67
95	262.80	262.24	0.21
100	266.40	265.35	0.40

Table 5.4: node3 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	193.30	192.84	0.24
5	195.85	196.42	0.29
10	200.59	200.00	0.29
15	201.13	203.58	1.22
20	205.70	207.16	0.71
25	211.59	210.74	0.41
30	213.67	214.31	0.30
35	220.74	217.89	1.29
40	221.36	221.47	0.05
45	225.67	225.05	0.27
50	231.25	228.63	1.13
55	233.75	232.21	0.66
60	232.08	235.79	1.60
65	240.25	239.37	0.37
70	241.46	242.95	0.62
75	248.38	246.53	0.75
80	248.31	250.10	0.72
85	252.99	253.68	0.27
90	257.86	257.26	0.23
95	261.11	260.84	0.10
100	264.13	264.42	0.11

Table 5.5: node4 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	197.51	196.99	0.26
5	201.69	200.86	0.41
10	205.09	204.74	0.17
15	206.97	208.61	0.79
20	212.08	212.49	0.19
25	216.78	216.36	0.20
30	220.02	220.23	0.10
35	219.90	224.11	1.91
40	227.79	227.98	0.08
45	232.60	231.86	0.32
50	241.49	235.73	2.39
55	237.70	239.60	0.80
60	244.67	243.49	0.49
65	248.23	247.35	0.35
70	248.83	251.23	0.96
75	256.22	255.10	0.44
80	259.04	258.97	0.03
85	262.72	262.85	0.05
90	266.26	266.72	0.17
95	270.90	270.60	0.11
100	273.75	274.47	0.26

Table 5.6: node5 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	250.62	249.17	0.58
5	254.81	253.37	0.56
10	259.70	257.58	0.82
15	261.01	261.78	0.30
20	261.86	265.99	1.58
25	272.81	270.19	0.96
30	275.56	274.40	0.42
35	278.55	278.60	0.02
40	283.31	282.81	0.18
45	286.77	287.01	0.08
50	290.64	291.22	0.20
55	293.73	295.42	0.58
60	297.66	299.62	0.66
65	306.58	303.83	0.90
70	307.68	308.03	0.11
75	311.61	312.24	0.20
80	318.26	316.44	0.57
85	322.22	320.65	0.49
90	325.85	324.85	0.31
95	330.65	329.06	0.48
100	333.47	333.26	0.06

Table 5.7: node6 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	199.75	200.60	0.43
5	202.64	204.12	0.73
10	206.39	207.65	0.61
15	211.87	211.17	0.33
20	215.31	214.69	0.29
25	226.13	218.22	3.50
30	220.47	221.74	0.57
35	225.03	225.26	0.10
40	227.63	228.79	0.51
45	233.01	232.31	0.30
50	237.00	235.84	0.49
55	234.98	239.36	1.87
60	243.48	242.88	0.24
65	243.92	246.41	1.02
70	249.02	249.93	0.36
75	253.22	253.45	0.09
80	256.63	256.98	0.13
85	260.08	260.50	0.16
90	265.96	264.02	0.73
95	268.14	267.55	0.22
100	271.91	271.07	0.31

Table 5.8: node7 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	200.75	198.79	0.98
5	204.79	202.56	1.09
10	208.25	206.33	0.92
15	209.99	210.10	0.05
20	211.14	213.87	1.29
25	206.85	217.64	5.21
30	223.95	221.41	1.13
35	226.67	225.18	0.66
40	229.83	228.95	0.38
45	232.54	232.72	0.07
50	238.09	236.49	0.68
55	240.02	240.25	0.10
60	242.42	244.02	0.66
65	251.87	247.79	1.62
70	251.25	251.56	0.12
75	257.03	255.33	0.66
80	258.34	259.10	0.29
85	262.06	262.87	0.31
90	264.68	266.64	0.74
95	271.30	270.41	0.33
100	274.37	274.18	0.07

Table 5.9: node8 Power Measurement

CPU Utilization(%)	Measured Power(W)	Calculated Power(W)	Error (%)
0	195.41	194.05	0.69
5	198.72	197.57	0.58
10	203.91	201.08	1.39
15	205.43	204.59	0.41
20	211.90	208.11	1.79
25	205.76	211.63	2.85
30	212.66	215.14	1.17
35	217.38	218.66	0.59
40	221.26	222.17	0.41
45	224.17	225.69	0.67
50	227.12	229.20	0.91
55	231.22	232.72	0.65
60	240.98	236.23	1.97
65	238.43	239.75	0.55
70	241.00	243.26	0.94
75	248.86	246.78	0.84
80	250.72	250.29	0.17
85	254.37	253.81	0.22
90	254.92	257.32	0.94
95	263.62	260.84	1.06
100	265.45	264.35	0.42

5.2 Throughput Modeling

We also model the throughput in relation to the cpu utilization. We define the throughput of our system to be the number of bytes processed per second in the map stage and it is measured in KB/s. Using the methodology described in section 3.2.1, we group our servers into 2 groups. This is because our testbed consists of 7 nodes with 4-core cpu and 1 node with 8-core cpu. We run our standard workload as given in Table 5.10 for different maximum utilization bound settings. We use an arrival rate such that all servers are utilized at their cpu utilization bounds during the map stage. We measure the throughput of the system during the time period where the system is running at a steady state. This is because when the workload is just submitted or when the system is at the ending stage of processing workload, the cpu utilization is lower than the bound. Therefore we ignore those transient periods at the starting and ending of the experiments.

We have logged the workload execution of our testbed with different utilization bound configurations. We then calculated throughput for these different scenarios. We used a modeling engine *Eureqa* [35] to model the relationship between the utilization bound and the throughput. Eureqa uses given data and relationship to create accurate predictive models using A.I. powered modeling engines.

Let u_{g_1} be the total utilization of the whole 8-core cpu group i.e. group1 and u_{g_2} be the total utilization of the whole 4-core cpu group i.e. group2. Then from the data collected, the total throughput τ was modeled to be a linear function given by:

$$\tau = 55.26u_{g_1} + 51.11u_{g_2} \tag{5.1}$$

Our model is of the form $\tau = a_1 u_{g_1} + a_2 u_{g_2}$.

5.3 Workload

In Hadoop YARN, the number of map tasks a job has depend upon both the input file size and the block size. Since the default block size is 128MB, we generated large files and uploaded them to HDFS. We distributed the file blocks uniformly in all our nodes. The generated files are of different sizes so that we can have MapReduce jobs with different number of map tasks. The replication factor used in our testbed is the default value of 3. We created our workload similar to that used in [36, 37] to evaluate our power management mechanism. This workload was generated from the distribution seen at Facebook over a week in October 2009 whose input sizes reflect the number of map tasks per job at that time [36]. Table 5.10 shows the job distribution at Facebook and in our testbed. We did not use the jobs with more than 200 map tasks because the total disk space in our small cluster restricted us from using very large files. We also reduced the number of 100-map and 200-map jobs in our workload for the same reason.

Table 5.10: Distribution of job sizes (in terms of number of map tasks at Facebook [36] and in our testbed)

Bin	#Maps in a Facebook job	% Jobs in the Facebook Trace	#Maps in a Benchmark job	#Jobs in our Benchmark
1	1	39%	1	38
2	2	16%	2	16
3	3-20	14%	10	14
4	21-60	9%	50	8
5	61-150	6%	100	3
6	151-300	6%	200	3
7	301-500	4%	N/A	0
8	501-1500	4%	N/A	0
9	$\geq$1500	3%	N/A	0

We used MapReduce wordcount application jobs as our workload. All together our workload contained 82 wordcount jobs with different number of map tasks. These jobs were distributed in the submission schedule with different lengths of interarrival time to achieve

different arrival rates as required by various experiments. We used the default FIFO scheduler in our cluster so that the jobs were run in the order of their submission. If the arrival rate was higher than that the cluster could process, jobs were placed in a queue and processed later following the order of their arrival.

5.4 Running MapReduce in Cgroup Containers

The cgroup containers were configured for cpu subsystem so that we could allocate cpu resources for our MapReduce tasks and control the maximum cpu utilized by them. To control MapReduce jobs to use only allocated resources, we need to run them in cgroup containers. First we need to create a cgroup hierarchy for cpu. We defined them in the cgroup configuration file named $cgconfig.conf$. We can create the hierarchy by mounting the cpu subsystem and then define the cgroups within this hierarchy. The resource allocation amount can be controlled from this cgroup definition.

```
mount {
    cpu = /cgroup/cpu;
}

group cgroup1{
    cpu {
        cpu.cfs_period_us = "1000000";
        cpu.cfs_quota_us  = "4000000";
    }
}
```

Figure 5.1: Example of cgconfig.conf file

Figure 5.1 shows a snippet of $cgconfig.conf$ file. The first 3 lines show *cpu* subsystem mounted in hierarchy named *cpu*. Then a cgroup named $cgroup1$ is created within which resource allocation is actually defined. This definition should be included in $cgconfig.conf$ file in every server in order for the whole cluster to use this $cgroup1$. For cpu subystem, as explained in section 2.2, we set the values for 2 parameters $cpu.cfs\_period\_us$ and $cpu.cfs\_quota\_us$.

The period is the total amount of time in microseconds available for the allocation per cpu core. We set this value to 1000000 i.e. 1 second. Since we have 4 core and 8 core cpus, to assign 100% of cpu time to this cgroup container, we need to set $cpu.cfs\_quota\_us$ to be 4000000 and 8000000 (period * number of cores) respectively. Similarly, we can set a different utilization bound for the container by choosing a different value for $cpu.cfs\_quota\_us$. In general, to assign x% to a container the quota value should be set to (period * number of cores * x%). The maximum value that can be assigned is the period and the minimum value is 1000 microseconds. Cgroup has the ability to tune the resource allocation in every server separately. That is, we can allocate different amounts of cpu resources for different servers by setting the quota to different values in different servers for the same cgroup.

We also need to define rules for Cgred service which moves the processes into cgroups [25]. We define that all processes belonging to the users in group *hadoop* use the resource allocation for *cpu* as defined in the cgroup $cgroup1$. This rule is added to the Cgred configuration file $cgrules.conf$ in all servers as:

@*hadoop cpu* $cgroup1$

We can define the rule for only a user $user1$ by replacing @*hadoop* by $user1$. If we replace it by $*$, then this rule is applied for all the users regardless of the group. The detailed implementation of cgroup and how to use it to allocate resources can be found in [25].

Now to run our workload within this cgroup container, we need to launch the process using *cgexec* command. The syntax is as given below:

cgexec -g subsystems : path-to-cgroup --sticky command arguments

Here *cgexec* is used to launch the command, the paramenter *-g* creates the process within the cgroup, *subsystems* is the resource, *path-to-cgroup* is the relative path for cgroup and *command* is the process command with *arguments*. The *--sticky* option before the task keeps any child process, if spawned, within the cgroup. This way, we can guarantee that the process does not use more cpu than the allocated amount. For instance the following command

runs the process *task* withing the cgroup named *cgroup*1 that controls cpu allocation.

*cgexec -g cpu : cgroup*1 *--sticky task*

In default hadoop implementation, the hadoop daemons resourcemanager, namenode, datanode and historyserver are started in the master node and nodemanager and datanode in slave nodes. When a job is submitted, these daemons monitor and allocate resources to the map and reduce tasks. To follow the cpu resource allocation as defined in our cgroup, we start all these hadoop daemons in cgroup too. For example, to start resourcemanage we use command:

*cgexec -g cpu : cgroup*1 *--sticky sbin/hadoop-daemon.sh start resourcemanager*

The initial part of the command "*cgexec -g cpu : cgroup*1 *--sticky*" is added before the default command to run the daemon within our defined cgroup *cgroup*1 that controls *cpu* resource allocation. Similarly, we need to submit our workload, the hadoop wordcount job, within our defined cgroup too.

*cgexec -g cpu : cgroup*1 *--sticky hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples*-2.5.2.*jar wordcount input output*

Here, the latter part of command is invoking *hadoop* to run jar file from path *share/hadoop/mapreduce/hadoop-mapreduce-examples*-2.5.2.*jar* for *wordcount* job with arguments *input* and *output*. When this job spawns maps and reduces in different servers, these tasks will also be executed in *cgroup*1. That is because all servers of the cluster have also defined this cgroup *cgroup*1. Depending upon the server a map or reduce task is run at, cpu usage can vary as each server may have set their own value for cpu resource allocation. Overall, the resource allocation amount is followed after these steps.

Chapter 6

Simulations and Experiments

In this chapter, we present simulations of algorithms that we introduced in Chapter 4. We also present results from our experiments and compare our simulation with our experimental results that we carried out in our testbed.

6.1 Simulations

We conducted several simulations to show the power savings that could be achieved using our PM technique. We also validated some of our simulation data in our real testbed using the workload described in section 5.3. Table 6.1 shows simulated power savings that can be obtained in our real testbed for different arrival rates (λ). The arrival rates that can be used in our PM policy for reducing total power consumption are constrained by our throughput model equation $\tau = 55.28u_{g1} + 51.1u_{g2}$. The lowest arrival rate is when all the servers utilization level is at 30% and the highest is when all the servers are fully utilized i.e. 100%. In addition to the throughput relation, the range of arrival rate that can be used also depends upon the number of servers in the system. For our testbed we chose λ between 12390 KB/s and 41298 KB/s as they are the lowest and highest possible arrival rates that

provided an opportunity for optimization.

Table 6.1: Simulation of power optimization in our testbed

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
13500	1806.78	1803.06	0.21
15000	1826.25	1823.82	0.13
20000	1904.13	1892.94	0.59
25000	1988.49	1961.76	1.34
28000	2033.92	2004.39	1.45
30000	2066.36	2035.38	1.50
35000	2144.24	2110.52	1.57
38000	2189.66	2163.84	1.18
40000	2222.11	2210.94	0.50

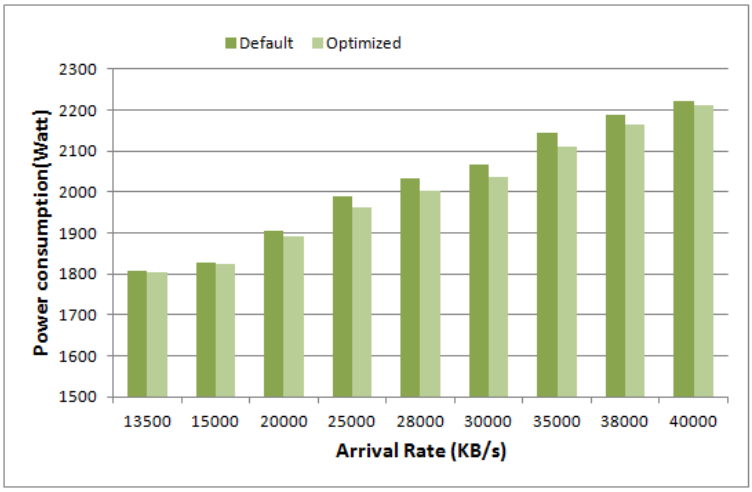

Figure 6.1: Comparison of total power consumption in our testbed for default vs optimized case

Table 6.1 shows the power consumption for the whole cluster in Watts for the given arrival rates with and without our PM optimization. For the default case, we assumed that all servers are utilized uniformly. For each arrival rate, we calculated the total utilization and distributed it uniformly among all servers. Then we calculated the total power consumption. For the optimized case, we used our optimization algorithm to calculate the utilization distribution

for a given arrival rate and then calculated the total power consumption. Figure 6.1 shows the comparison of simulated default vs simulated optimized total power consumption for our testbed.

Here the maximally achieved reduction in total power consumption is approximately 1.6%. The reason behind it might be the small size and relatively less heterogeneity of our testbed. Therefore, we conducted further simulations using synthetic clusters. We considered a heterogeneous cluster that is divided into two groups consisting of 10 servers in group1 and 70 servers in group2. We increased the number of servers in each group in the same ratio i.e. by 10 times. We assumed that the cluster consisted of multiple servers equivalent of our real testbed servers with same power equations as given by Table 5.1 and same throughput models. With the increased size, we also adjusted the arrival rates for the cluster as larger cluster can process more data. The maximum power consumption reduced is nearly 1.8%. Table 6.2 shows the total power optimization that can be achieved in this environment.

Table 6.2: Simulation of power optimization for a cluster containing 10 servers in group1 and 70 servers in group2

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
150000	17400.10	17364.57	0.20
200000	18191.37	18080.10	0.61
250000	18982.64	18776.15	1.09
280000	19444.21	19188.25	1.32
300000	19773.91	19462.27	1.58
350000	20565.18	20212.10	1.72
380000	21092.69	20724.70	1.74
400000	21356.44	21267.70	0.42

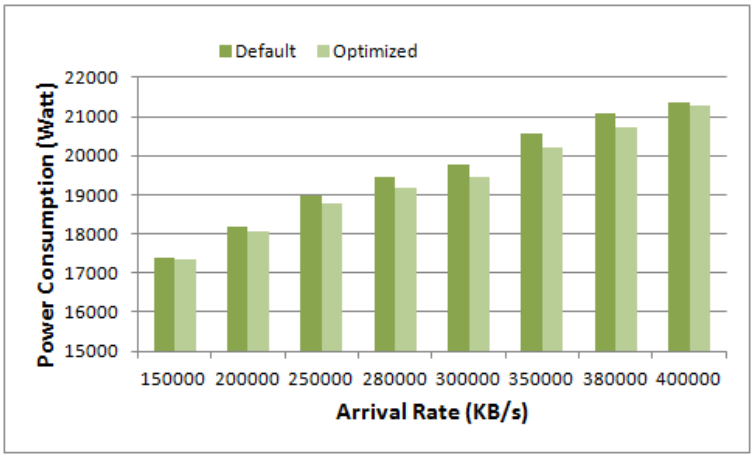

Figure 6.2: Graphical comparison of power optimization as given by Table 6.2

We further changed the testbed to contain the same number of servers in each group while keeping the power equation and throughput equations same. In this case, our simulations show that the power consumption reduction is increased. The total power consumption is reduced by up to approximately 3%. Table 6.3 shows the total power optimization that can be achieved in this setting for different arrival rates.

Table 6.3: Simulation of power optimization for a cluster containing 2 groups with 70 servers each

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
250000	30548.72	30493.76	0.18
300000	31547.14	31183.37	1.15
350000	32545.56	31870.93	2.07
400000	33401.34	32606.11	2.38
450000	34399.76	33364.70	3.01
500000	35398.18	34370.67	2.90
550000	36253.96	35543.19	1.96
600000	37252.38	36714.41	1.44
650000	38250.80	37886.93	0.95
700000	39249.21	39058.16	0.49

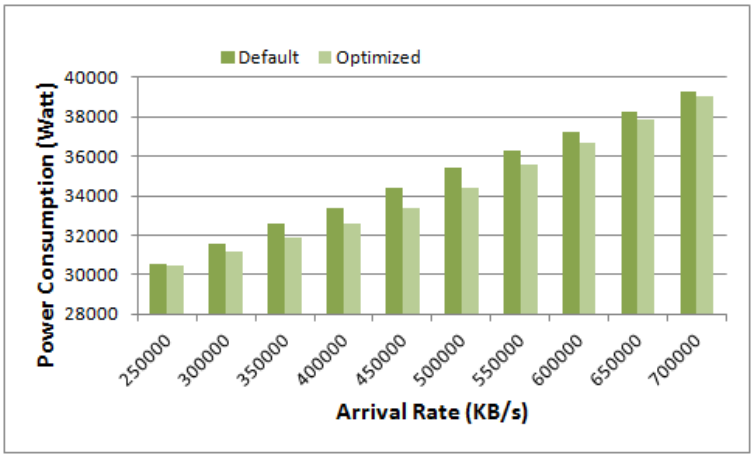

Figure 6.3: Graphical comparison of power optimization as given by Table 6.3

We also conducted simulations by changing the power relation in the servers. We used 70 servers in each group and assumed the throughput relation to be the same but changed the power relation of some of the servers to make the server more power inefficient i.e. incremented the value of α in the power relation given by Equation (3.2). From our simulations we could observe that the more power efficient servers give more opportunities for power optimization. The result of power optimization in this setting is as given in Table 6.4.

Table 6.4: Simulation of power optimization for a cluster with more power efficient servers divided into 2 groups containing 70 servers each

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
250000	30091.19	29993.45	0.32
300000	31183.46	30683.59	1.60
350000	32275.73	31384.22	2.76
400000	33211.97	32093.99	3.37
450000	34304.24	32844.48	4.26
500000	35396.51	33906.80	4.21
550000	36332.75	35264.30	2.94
600000	37425.02	36620.30	2.15
650000	38517.29	37977.80	1.40
700000	39609.57	39333.80	0.70

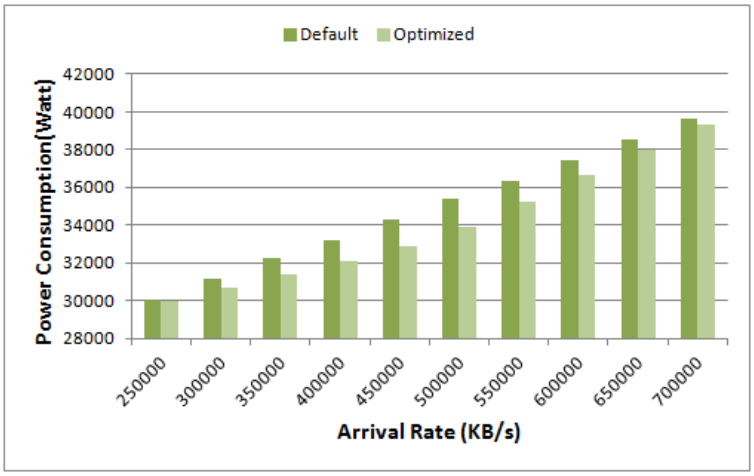

Figure 6.4: Graphical comparison of power optimization as given by Table 6.4

We also studied the effect of throughput on the power optimization of the cluster. We changed the throughput of the cluster to analyze its effect on the utilization distribution and hence total power consumption. We changed the throughput relation of the cluster to be $\lambda = 31.25u_{g1} + 53.05u_{g2}$. We assumed the servers to be divided into 2 groups with 70 servers each. We also changed the power relation of the servers to contain more power efficient servers. We observed that our optimization strategy works better in such cluster. In this setting, the total power consumption can be reduced up to 8.5%. From this result we can say that the power optimization is maximized when the power efficient servers also have more throughput capacity. The optimization in such a setting for different arrival rates are given in Table 6.5 below.

Table 6.5: Simulation of power optimization for a cluster containing 2 groups with 70 servers each with throughput = $31.25u_{g1} + 53.05u_{g2}$

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
250000	31491.34	30613.47	2.79
280000	32271.03	31019.24	3.88
300000	32738.85	31286.30	4.44
330000	33518.55	31684.02	5.47
350000	34142.30	31948.22	6.43
380000	34922.00	32368.29	7.31
400000	35389.81	32660.33	7.71
430000	36169.51	33087.04	8.52
450000	36793.26	33811.30	8.10
480000	37572.96	35251.30	6.18
500000	38040.78	36211.30	4.81
530000	38820.47	37651.30	3.01
550000	39444.23	38611.30	2.11

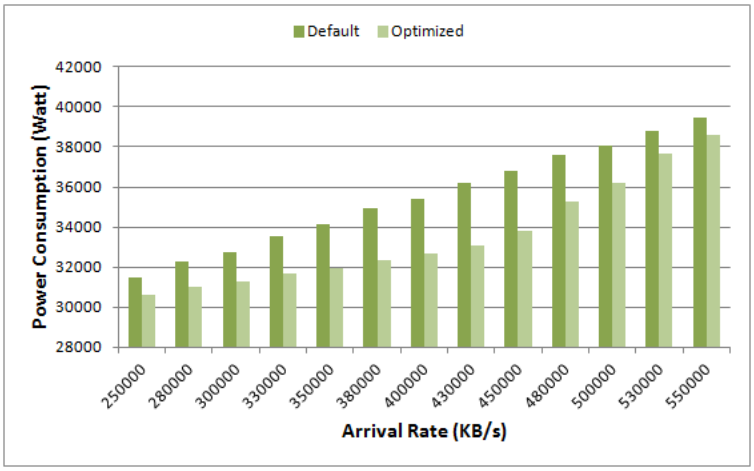

Figure 6.5: Graphical comparison of power optimization as given by Table 6.5

We further simulated the cluster environment for servers divided into 3 groups and calculated the power reduction for different arrival rates. We assumed the cluster to contain 70 servers in each group. We also assumed the throughput relation in such cluster to be throughput = $31.25u_{g1} + 53.05u_{g2} + 75.35u_{g3}$. We assumed the system to contain the servers similar to our testbed with similar power relations. The following table 6.6 shows our result

for this setting.

Table 6.6: Simulation of power optimization for a cluster containing 3 groups with 70 servers each with throughput = $31.25u_{g1} + 53.05u_{g2} + 75.35u_{g3}$

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
350000	44635.75	44565.68	0.16
400000	45965.64	45249.07	1.56
450000	47029.56	45927.04	2.34
500000	48359.46	46589.29	3.66
550000	49423.37	47290.00	4.32
600000	50753.27	48166.50	5.10
650000	51817.18	50566.50	2.41
700000	53147.08	51979.82	2.10
750000	54210.99	52683.66	2.82
800000	55540.89	53371.53	3.91
850000	56604.80	54038.28	4.53
900000	57934.70	54712.32	5.56
950000	58998.61	55433.99	6.04
1000000	60062.53	57346.80	4.52
1050000	61392.43	59746.80	2.68
1100000	62456.34	62146.80	0.49

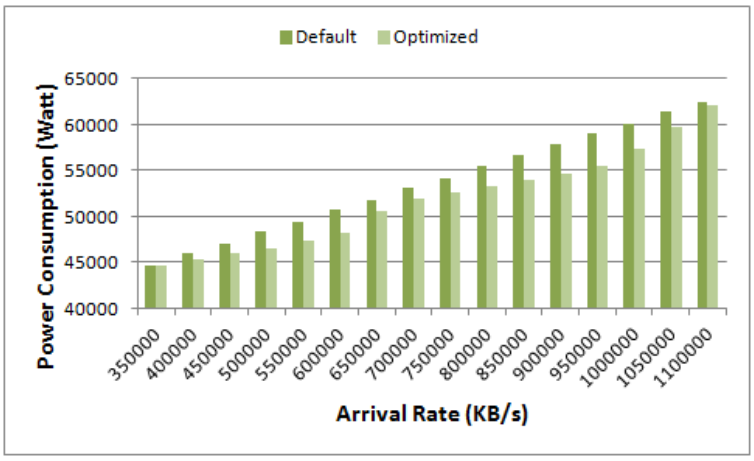

Figure 6.6: Graphical comparison of power optimization as given by Table 6.6

6.2 Experimental Results

Table 6.7 shows the results of running the experiments in our testbed for different arrival rates. From the table, we can observe that the maximum power savings is achieved for $\lambda = 28000KB/s$ and $\lambda = 30000KB/s$. For default case, we ran the experiment without using any cgroup container and collected the data. For optimized case, we ran our experiments in cgroup containers with cpu resource allocation as calculated from our optimization algorithms and collected the data.

Table 6.7: Experimental Power savings for our testbed

Arrival Rate(KB/s)	Default Power(W)	Optimized Power(W)	% power saved
13500	1750.73	1744.63	0.35
15000	1768.70	1760.07	0.49
20000	1830.45	1819.89	0.58
25000	1888.68	1868.58	1.06
28000	1920.75	1892.45	1.47
30000	1953.44	1922.06	1.61
35000	2005.24	1985.67	0.98
38000	2049.84	2032.37	0.85
40000	2073.63	2072.10	0.07

We present this data graphically in Figure 6.7 below.

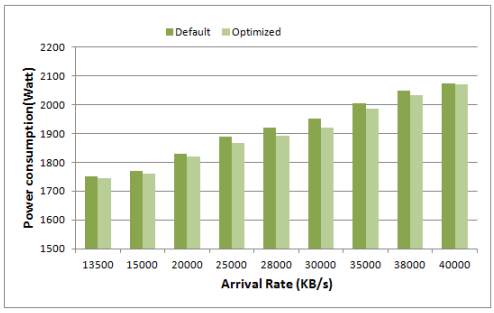

Figure 6.7: Comparison of optimized power during our experiments versus default power consumption

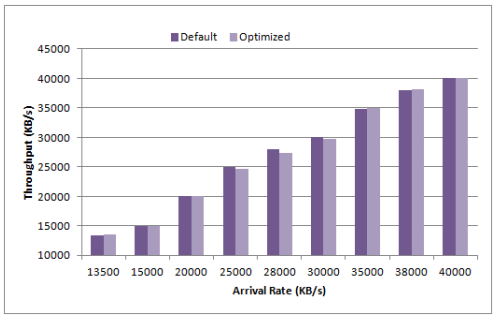

Figure 6.8: Comparison of actual throughput obtained in our experiments versus expected throughput

Figure 6.8 shows that the experimental throughput is almost same as the expected throughput. It shows that our optimization process seeks to reduce total power consumption while maintaining the expected throughput. The throughput is always withing ±1% of the expected value. Our optimization process has very minimal effect on the throughput of the cluster. The system consumes more power to produce higher throughput. So, we agree that if a system does not have rigid throughput limitations and allows some degradation in throughput, then the power saving can be more significant.

Comparing our experiments with our simulations, we find that the power consumption value is higher during simulations than the experimental values as shown in Figure 6.9. The optimized total power consumption obtained from the experiments is on average approximately 5% lower than the optimized total power consumption expected from the simulations.

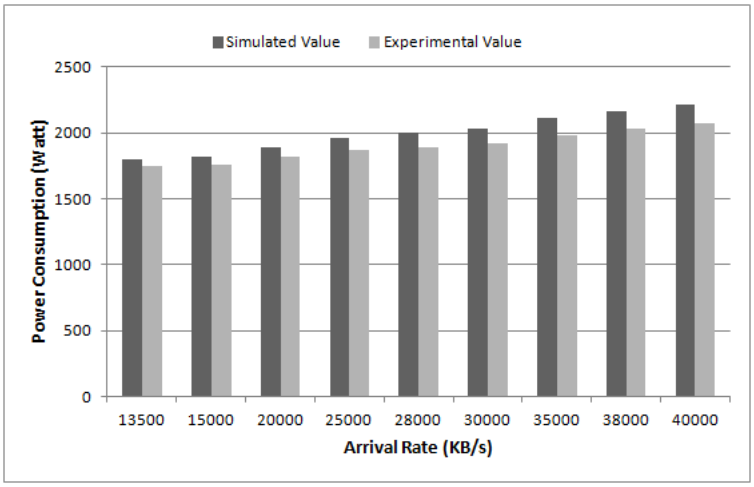

Figure 6.9: Comparison of total power consumption in our experiment versus our simulation

This is because we set the value for utilization bound only, the server may be utilized less than the bound. When calculating the default value, we consider that the server is utilized at the same level while processing the whole workload. But in actual case, the server is not utilized at the bound level all the time. We have Hadoop MapReduce jobs as our workload

and the map tasks are more computation intensive while the reduce tasks are not. So when a server is processing a reduce task, its cpu utilization will be lower than the allocated value. Therefore, we observe the difference in actual value versus the expected value from our simulation.

The comparison of the power saving percent in our simulated environment versus the experimental result is shown in Figure 6.10 below. These profiles of the power consumption improvement expressed in percent are somewhat similar.

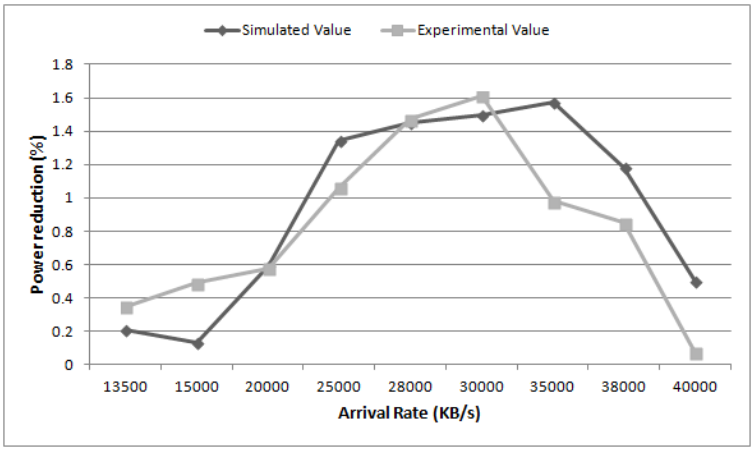

Figure 6.10: Comparison of total power reduction percent in our simulation versus our experiment

Chapter 7

Conclusion

In this thesis, we investigated a power management policy in a Heterogeneous MapReduce cluster. We developed an optimization algorithm based on the fact that the servers are utilized less than their capacity most of the time. The essence of our algorithm was to utilize the servers intelligently so that the total power consumption of the cluster was optimized. We first group our servers into different groups based on their capacity and develop the power model and throughput model of these groups. We then use these models in our proposed optimization strategy. Our optimization policy works in two steps. The first step calculates the usage requirement for the given load or arrival rate. In second step, we adopt a greedy approach in distributing this resulting utilization to different servers in the group based on their power efficiency. In our policy, we allocate higher usage to power efficient servers than the inefficient ones.

We then examine the effectiveness of this policy as it is applied to our testbed and find that the total power consumption has been reduced when it is applied. We have further explored and studied the optimization by simulating it in different scenarios. We have found that the algorithm works well when the system is more heterogeneous i.e. when the power consumption and throughput profile of the servers are dissimilar from each other.

Chapter 8

Future Work

In this work, our experiments show a small improvement in total power consumption of the cluster due to its small size and less heterogeneity. When we simulated the power optimization for large heterogeneous clusters to explore the potential of our algorithm, we observed significant improvements. In the future, we can actually verify this by applying our algorithm to a real heterogeneous cluster with a larger number of different servers. In our experiments, we used MapReduce wordcount jobs as our workload which read from files and count the words in that file and write the results to output files. Hence the nature of our workload is not only cpu intensive but also I/O intensive. We can extend this to different types of workloads in the future and analyze its effects. Another interesting work can be to study the power optimization when our algorithm is applied in combination with the DVFS method with data placement strategies. We studied the power consumption based on the cpu usage and did not study the effects of other components like disks, memory, switches, fans etc. To make our PM strategy more robust, the effect of power consumption by these components could be included in the future.

Bibliography

[1] Natural Resources Defense Council. Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. http://www.nrdc.org/energy/files/data-center-efficiency-assessment-ip.pdf. 2014.

[2] ASHRAE Technical COmmittee 99. Datacom equipment power trends and cooling applications 2005.

[3] C. Belady. "In the data center, power and cooling costs more than the equipment it supports". February. URL http://www.electronics-cooling.com/2007/02/in-the-data-center-power-and-cooling-costs-more-than-the-it-equipment-it-supports/.

[4] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller. "Energy Management for Commercial Servers". *IEEE Computing*, vol. 36(12):39–48, December 2003.

[5] L. Wang and Y. Lu. "An Efficient Threshold-Based Power Management Mechanism for Heterogeneous Soft Real-Time Clusters". *IEEE Transactions on Industrial Informatics*, vol. 6(3):352–364, August 2010.

[6] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Zhijian Lu. "Power-aware QoS management in Web servers". In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 63–72, December 2003.

[7] S. Ibrahim, D. Moise, H.-E. Chihoub, A. Carpen-Amarie, L. Bougé, and G. Antoniu. "Towards Efficient Power Management in MapReduce: Investigation of CPU-Frequencies Scaling on Power Efficiency in Hadoop". 2014.

[8] T. Wirtz and R. Ge. "Improving MapReduce energy efficiency for computation intensive workloads". In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, July 2011.

[9] J. Leverich and C. Kozyrakis. "On the Energy (In)efficiency of Hadoop Clusters". March 2010.

[10] R.T. Kaushik and M. Bhandarkar. "GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster". In *Proceedings of the 2010 international conference on Power Aware Computing and Systems*, pages 1–9, January 2010.

[11] X. Fan, W.-D. Weber, and L.A. Barroso. "Power Provisioning for a Warehouse-sized Computer". In *ISCA: Proceedings of the 34th annual international symposium on Computer architecture*. ACM, 2007.

[12] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. "Optimizing Job Performance Under a Given Power Constraint in HPC Centers". In *Green Computing Conference, 2010 International*, pages 257–267, August 2010.

[13] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. "Optimal Power Allocation in Server Farms". *SIGMETRICS/Permormance*, June 2009.

[14] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose, and M. Martonosi. "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a

Given Power Budget". In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 347–358, December 2006.

[15] J. Son and R. Buyya. "SLA-aware and Energy-efficient Dynamic Overbooking in SDN-Enabled Cloud Data Centers". 2016.

[16] "New data supports finding that 30 percent of servers are Comatose, indicating that nearly a third of capital in enterprise data centers is wasted", June 2015. `http://tsologic.com/wp-content/uploads/2015/06/AnthesisGroup_Case-Study_30PercentComatose_06032015.pdf`.

[17] Anthesis group. `www.anthesisgroup.com`.

[18] L. A. Barroso and U. Hölzle. "The Case for Energy-Proportional Computing". *IEEE Computer*, vol. 40:33–37, 2007.

[19] G. Linden. "Make Data Useful". 2006. `http://www.gduchamp.com/media/StanfordDataMining.2006-11-28.pdf`.

[20] L. Minas and B. Ellison. "The Problem of Power Consumption in Servers". 2009.

[21] The Apache Hadoop. Apache Hadoop documentation, 2016 : `http://hadoop.apache.org/`.

[22] HDFS documentation. `http://hadoop.apache.org/docs/r2.5.2/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html`.

[23] HADOOPTPOINT. `http://www.hadooptpoint.com/hadoop-distributed-file-system/`.

[24] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media/Yahoo Press, 3rd edition, May 2012.

[25] P. Ondrejka, M. Prpiĉ, R. Landmann, and D. Silas. "Red Hat Enterprise Linux 6 Resource Management Guide". 2013.

[26] Dynamic frequency scaling. `https://en.wikipedia.org/wiki/Dynamic_frequency_scaling`.

[27] Dynamic voltage scaling. `https://en.wikipedia.org/wiki/Dynamic_voltage_scaling`.

[28] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. "Energy-Efficient Real-Time Heterogeneous Server Clusters". In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 418–428, April 2006.

[29] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis". In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 43–56, 2012.

[30] Í. Goiri, K. Le, T.D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. "GreenHadoop: leveraging green energy in data-processing frameworks". In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 57–70, 2012.

[31] S. Li, T. Abdelzaher, and M. Yuan. "TAPA: Temperature Aware Power Allocation in Data Center with Map-Reduce". In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, July 2011.

[32] W. Lang and J.M. Patel. "Energy Management for MapReduce Clusters". *SIGOPS Oper. syst. Rev.*, 44:61–65, March 2010.

[33] N. Yigitbasi, K. Datta, N. Jain, and T. Willke. "Energy Efficient Scheduling of MapReduce Workloads on Heterogeneous Clusters". December 2011.

[34] The Apache Hadoop YARN. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[35] Eureqa. http://www.nutonian.com.

[36] M. Zaharia et al. "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling". *EuroSys*, 2010.

[37] C. He, Y. Lu, and D. Swanson. "Matchmaking: A New MapReduce Scheduling Technique". In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 40–47, November 2011.

[38] Q. Zheng and B. Veeravalli. "Utilization-based pricing for power management and profit optimization in data centers". *Journal of Parallel and Distributed Computing*, pages 27–34, January 2012.