SCHEDULING FOR CPU PACKING AND NODE SHUTDOWN TO REDUCE THE ENERGY CONSUMPTION OF HIGH PERFORMANCE COMPUTING CENTERS

By

Srikanth Phani Vudayagiri

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Computer Science

2010

ABSTRACT

SCHEDULING FOR CPU PACKING AND NODE SHUTDOWN TO REDUCE THE ENERGY CONSUMPTION OF HIGH PERFORMANCE COMPUTING CENTERS

By

Srikanth Phani Vudayagiri

During the past decade, there has been a tremendous growth in the high performance computing and data center arenas. The huge energy requirements in these sectors have prompted researchers to investigate possible ways to reduce their energy consumption. Reducing the energy consumption is not only beneficial to an organization economically but also to the environment. In this thesis, we focus our attention on high performance scientific computing clusters. We first perform experiments with the CPU Packing feature available in Linux using programs from the SPEC CPU2000 suite. We then look at an energy-aware scheduling algorithm for the cluster that assumes that CPU Packing is enabled on all the nodes. Using simulations, we compare the scheduling done by this algorithm to that done by the existing, commercial Moab scheduler in the cluster. We experiment with the Moab Green Computing feature and based on our observations, we implement the shutdown mechanism used by Moab in our simulations. Our results show that Moab Green Computing could provide about an 13% energy savings on average for the HPC cluster without any noticeable decrease in the performance of jobs.

To,

My parents, brother and sister.

ACKNOWLEDGEMENT

First of all, I'd like to thank my parents, brother and sister for their unconditional love and support, and for encouraging me to pursue a graduate degree in USA.

I owe my deepest gratitude to Dr. Richard Enbody, Dr. Eric Torng and Dr. Bill Punch for their patience, guidance and support throughout my MS.

I am thankful to all the Professors in the CSE and Math department with whom I took courses. The quality of their lectures and their emphasis on understanding have contributed immensely to my growth professionally.

Special thanks to my mentor and friend, Jignesh Patel, for sharing his thoughts and insights into topics related to computer science, physics, mathematics, astronomy, evolution, (...) and for his continuous encouragement.

Additionally, I would like to thank:

The students of CSE-231 for being patient and for giving me a good teaching experience.

Alok Watve for being a supportive friend.

Andy Keene and Jim Leikert, from the MSU HPC Center, for their cooperation and for providing the log files. Thanks to Jim again for introducing me to the Hanayama Cast Puzzles.

Nick Ihli from Adaptive Computing for providing a lot of help related to Moab.

Jessica Moy and Joel Lenz from Department of RS & GIS, MSU for the graduate assistantship during Fall '09.

Linda Moore and Norma Teague for helping me with all the administrative issues.

Brooke Stokdyk, from the OISS, for her great advice on numerous occasions.

The beautiful MSU campus for providing a peaceful and invigorating environment much needed for education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

As of 2010, many universities, research organizations and corporations across the world run their own data centers using clusters of computers. Some data centers are used for scientific computing (high performance computing centers) whereas others are used for commercial purposes (e.g. web hosting centers) to provide services to customers. Regardless of their purpose, the common denominator to all data centers is the huge energy demand required for operating and cooling the infrastructure. With energy at a premium and governments encouraging everyone to go green, the research community has put in a lot of effort during the past decade to find ways to reduce the overall energy consumption of data centers. In this thesis, we focus specifically on reducing the energy consumption of high performance computing centers.

1.1 Need for energy efficiency in clusters

There are three main reasons why we desire that clusters be energy efficient:

1. **Scarcity of Power**

   According to a report released by the EPA in 2007 [12], 1.5% of all electricity generated in US during 2006 was consumed by data centers alone and it's expected to double by 2011. Furthermore, the report states that an additional 10 power plants would be needed to meet the new demands. This result not only indicates that the power consumption of data centers is very high but also that at this rate of consumption, eventually there won't be enough power to operate all of them at full capacity.

2. **High Operating and Cooling Costs**

   Buying the infrastructure for a cluster is a one-time investment (with perhaps an annual

support contract), but the demand for energy required to operate and cool the nodes persists for the entire life-time of the nodes. Estimates show that the energy cost for a server surpasses the procurement cost long before it is decommissioned [8]. As data centers usually consist of hundreds of nodes, energy costs are a major contributor to the total cost of ownership.

3. **Harmful Effects On the Environment**

   There is a problem in our usage of non-renewable energy sources like coal, natural gas, nuclear power and so on that overwhelmingly dominate power generation today. In the US, approximately 85-90% of the power generated comes from non-renewable sources [11, 36]. Usage of these sources pollutes the environment with emissions of CO_2 and other greenhouse gases. As the data center sector has a huge energy consumption (as noted above), reducing the energy demand of data centers would help in reducing the need for power generation (the supply) which in turn would reduce pollution.

1.2 Terminology

Before proceeding any further, we define the terms used in the text.

1. Core: The basic computing element capable of running at most one job at a time.
2. Multi-core Processor or Chip Multi-Processor (CMP): Computing device that has at least two cores. In this text, we refer to it as simply the processor or CMP. Note that Moab scheduler's definition of a processor is equivalent to our definition of a core.
3. Idle Processor: All the cores on the processor are idle. Equivalently, no job is running on the processor.
4. Busy Processor: At least one core on the processor is running a user job.
5. Node (or Machine or Server): Hardware with one or more processors, some memory and a disk capable of running one or more user jobs. Note: Hereafter, the terms node, machine and server all have the same meaning.

6. Idle Node: The node is booted up and running essential OS services, but not running any user jobs (i.e. all the processors are idle).
7. Busy (or Active) Node: The node is booted up, running essential OS services and running at least one user job requiring one or more dedicated cores, some memory and disk.
8. Blade: A physical case that hosts one or more nodes.
9. Cluster: A collection of homogeneous nodes that are connected using a local high-speed, high-bandwidth network.
10. Job: A program that requests exclusive access to one or more cores, memory and disk space on one or more nodes for a specific time duration (known as the *walltime*).
11. Walltime: The time duration requested by a job. Note that the walltime is just an estimate of the runtime of a job provided by the user.
12. Task: In Moab terminology, a task is a logical unit consisting of a single node with some number of dedicated cores, some memory and some disk space within the node. If a job requests for N nodes, it is in effect requesting for N tasks. A lot of times, it's convenient to talk in terms of tasks rather than nodes, cores and memory.
13. Queue: A logical container for the jobs submitted by the users. It is also known as a "class" in Moab terminology.
14. Resource manager (e.g. Torque): Software that manages and keeps track of the queue, the jobs and nodes of the cluster.
15. Scheduler (e.g. Moab): Software that makes the intelligent decisions, based on some criteria, about where the jobs in the queue should run and instructs the resource manager to take the necessary steps to start the job.

1.3 Model of our HPC cluster

Figure 1.1 shows a schematic representation of an HPC cluster. The various components of the cluster are listed below:

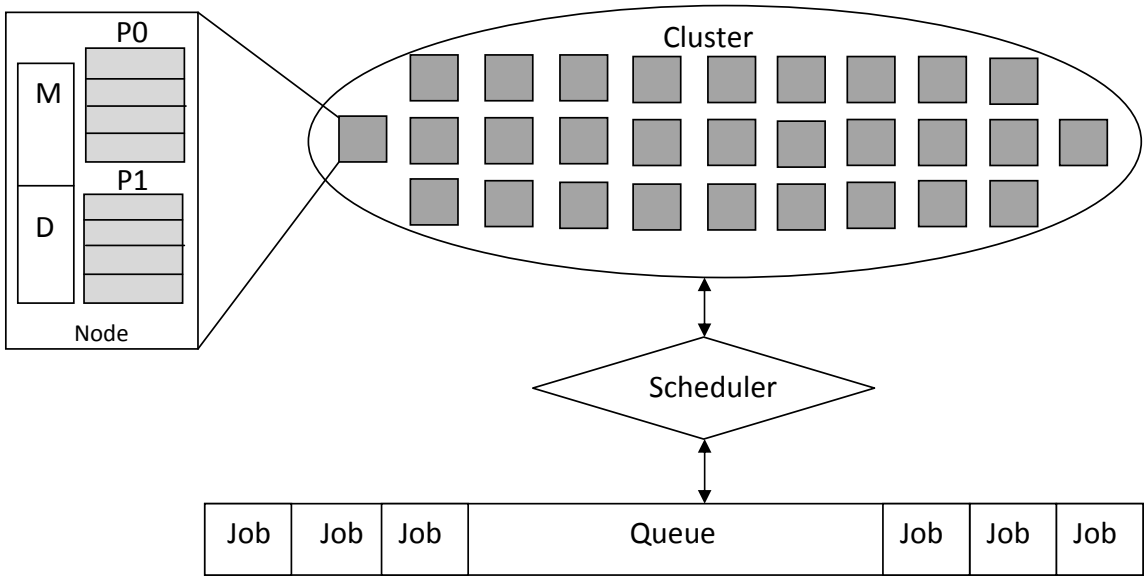

Figure 1.1: Schematic model of HPC Cluster

1. There are N identical nodes in the cluster. In the figure, N=29.
2. Each node has P CMPs with c cores per CMP (dark gray boxes), M GB memory and D GB disk. In the figure, P=2, c=4. The CMPs are shown as $P0$ and $P1$.
3. The cluster has a centralized scheduler.
4. The cluster has a single queue to which the users submit jobs.
5. Each job j in the cluster requests $R(j)$ nodes, $C(j)$ cores per node, $M(j)$ amount of memory, $D(j)$ amount of disk space and a walltime of $W(j)$ to complete its task. The *walltime* is the duration for which resources would be reserved for the job once it starts running. All the jobs are non-preemptive and independent of each other. Jobs started on a node cannot be migrated to another node.

Many scientific computing clusters operate using this model and employ a commercial scheduler such as Moab [4] or Load-Leveler [3] to do the scheduling.

1.4 Motivation

CPU Packing and Node Shutdown are two of the three available techniques to save energy, the third one being dynamic voltage and frequency scaling (DVFS). While CPU Packing and DVFS have the potential to save energy within a node using software and hardware support, node shutdown technique saves energy by simply turning off the nodes when not needed.

"CPU Packing" simply means to pack the available jobs on the cores of one CMP and once all of its cores are busy, pack another CMP on the same node and so on. This technique is possible only on nodes that have multiple CMPs (in other words, an SMP system having more than one multi-core processors). Since each node in the MSU HPC cluster has two CMPs, this technique is viable.

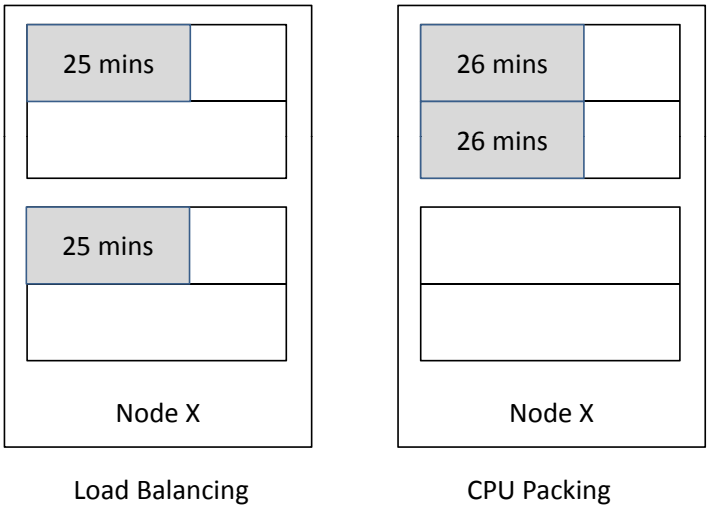

Figure 1.2: CPU Packing motivation example

While experimenting with the power management features on a test node consisting of two dual-core CMPs, two instances of the Vortex program belonging to the SPEC CPU2000 [6] integer benchmarks suite were started simultaneously on the node. The Vortex program performs database transactions and uses about 50 MB of memory. If the two instances are packed onto a CMP, the power consumption of the node is 141.5 Watts while the runtimes of the instances are approximately 26 minutes. On the other hand, if each instance is run on a different CMP, the power consumption of the node increases to 148 W and the runtimes

of the instances are approximately 25 minutes. The difference (148*25 - 141.5*26) = 21 Watt-mins is the total energy savings for running the same job with CPU Packing. There is also an additional 120 Watt-mins energy consumed by the node during the 26th minute in the case of jobs not being packed (assuming the idle node power consumption of the test node to be 120 Watts and that no programs were run on the node after the Vortex instances completed execution). For some other programs belonging to the suite (e.g. Crafty), CPU Packing showed even more energy savings. This observation was the main motivation to investigate the CPU Packing technique. Figure 1.2 shows Packing vs. Load Balancing.

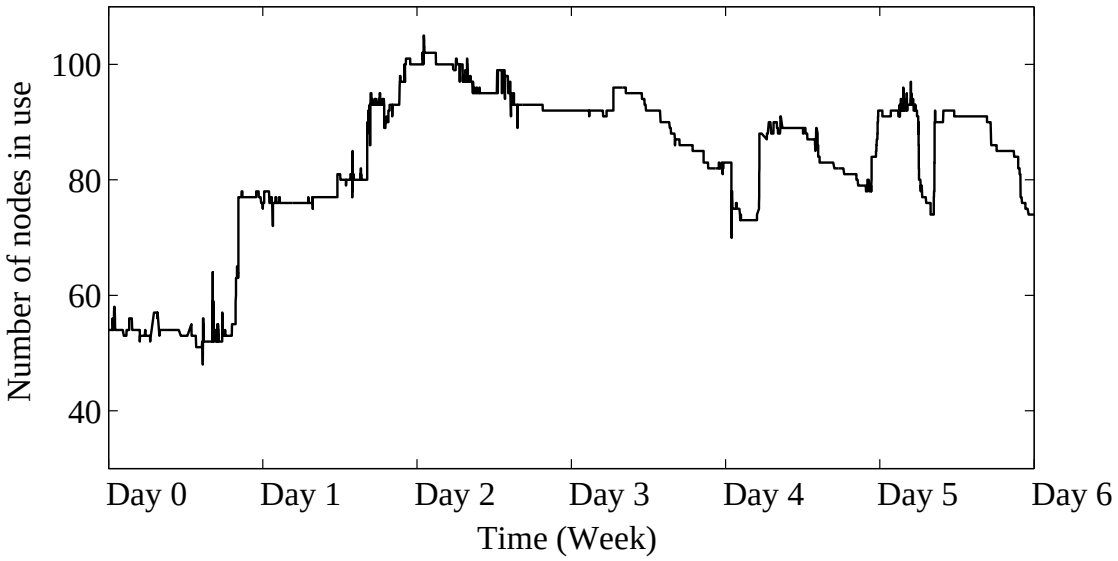

Figure 1.3: Node shutdown motivation example

Although the above approach looks promising, it provides only nominal energy savings in practice.

Figure 1.3 shows the node usage in the cluster during a week. It's clear that there are periods during the week when the load on the cluster is less and the number of busy nodes in the cluster is much less than the maximum number of nodes available. Since a single node in the HPC cluster consumes approximately 220 W, its possible to achieve substantial savings by shutting down the idle nodes during those periods of less traffic.

Note that the foremost objective of our research was to do experimentation and analysis

that has practical value to the HPC Cluster Administrators and something that could be implemented in the cluster. We strongly believe that CPU Packing and Node Shutdown techniques can be used in practice to save energy.

1.5 Thesis Outline

Chapter 2 is dedicated to the related work and briefly describes the contemporary approaches for power management. Chapter 3 provides details of the existing power-related features in hardware and software and our initial investigations. The results in this chapter form the basis for our research in the next chapter. Chapter 4 explains our research in detail, describes the simulations and presents the results. The final chapter gives the conclusions and future work.

Chapter 2

Related Work

From a general engineering perspective, an energy efficient cluster would require, among other things, efficient power supply and cooling units, accurate sensors and a well-designed server room. However, these objectives lie beyond the scope of computer science. Computer science deals specifically with the power consumption of the servers within a cluster; and using the available technologies, there really are only three possible ways to reduce the power. Naturally, all the existing research could be classified based on these approaches (or their combinations).

2.1 Packing jobs on CMPs (CPU Packing)

2.1.1 Idea

On a node having multiple CMPs (most server motherboards support two), it might be more energy efficient to pack the jobs on a minimum number of CMPs while the other CMPs are turned off by the operating system thereby saving energy. We have already seen an example in Chapter 1.

As more jobs are packed on a CMP (the maximum being the number of cores available), the performance of each job running on the CMP might degrade depending upon whether the job is CPU-intensive or memory-intensive and whether there are shared resources among the cores on the CMP (shared caches, FSB etc.).

2.1.2 Literature

Ghiasi et al. [22] experiment with CPU Packing on a Symmetric Multi-Processor (SMP) system consisting of multiple CMPs and using web-based workloads. They study four policies (namely, Oracle, PAST, AVG <N> and Hybrid) that estimate the utilization of the SMP system during a future interval using data from previous sampling intervals. The policies determine the number of processors that would be needed to handle the load allowing the other processors to be transitioned into low power state. They report energy savings of about 25% with a performance reduction of only about 3% using their hybrid policy.

Freeh et al. [19] investigated a combined model of DVFS (see below) and CPU Packing on a cluster of eighteen nodes with each node having two dual-core AMD Opteron CMPs. For this model, they evaluate the performance of various parallel HPC applications with different CPU frequencies and packing configurations. Their results indicate that for HPC workloads, packing on a single core on multiple nodes provides no energy savings whereas packing on two or more cores may or may not provide energy savings depending on the HPC application. Our experiments, given in the next chapter, confirm the result for the case of two cores on a single node. For web based commercial workloads, their experiments show that packing could provide upto 19% energy savings.

Suresh Siddha et al. [33] describe the processor packing feature developed by them for the Linux kernel. They show that packing reduces power depending upon the job profile and the resources shared (cache, FSB etc.) by the jobs. The same authors [34] present their findings about the effect of packing on the performance of different programs (SPECjbb2005, SPECjbb2000, SPECfp_rate, IMDS). They consider various CMPs with different sizes of shared caches (L2) among the cores and evaluate the performance of the programs with and without packing on the CMPs. Their experiments show that for an L2 cache size of 16MB or more, CPU Packing does not cause performance degradation for any of the jobs considered.

2.2 Node Shutdown

2.2.1 Idea

Medium to large size clusters consist of hundreds of nodes and at any given time, all the nodes might not be processing jobs. Keeping the nodes turned on during long periods even when there is no work to do wastes energy. Therefore, when the load on the cluster is low, some of the nodes may be shutdown and restarted later whenever the load increases.

Advanced Configuration and Power Interface [1], commonly known as ACPI, defines different sleep states such as standby, suspend and shutdown for servers (refer to section on ACPI in chapter 3). All ACPI-compliant servers implement these states for different levels of shutdown.

The only potential drawback of this approach is due to the fact that the cluster scheduler does not have any knowledge about future jobs. If the decision to shutdown some nodes turns out to be wrong (i.e. some new jobs need to be run immediately and can't be run on the current set of active nodes), it might be necessary to immediately follow up the shutdown of nodes by restarting them and in the process consume more energy than if they were to remain idle during that time. On the other hand, if a correct decision was made, its guaranteed to give valuable savings.

2.2.2 Literature

Pinheiro et al. [31, 30] use a control theoretic approach to shutdown or restart nodes in a cluster depending upon the current and predicted load. They allow migration of the jobs from the nodes to be shutdown to other active nodes.

Chase et al. [15] investigate the energy consumption of Internet Hosting Centers. These centers usually host multiple websites and all the provisioning resources are shared among those sites. Using dynamic resource management, the authors maintain a minimal set of servers that could handle the collective load for all the websites while shutting down the

remaining servers.

Rocks is a Linux software distribution targeted towards clusters. Cluster-powersave is a utility program available for a Rocks cluster and it uses a very simple approach to shutdown and wake up nodes. Whenever there is a job waiting in the queue, the scheduler will wake up enough nodes so that the job could be started. If there are no jobs in the queue, it checks for any idle nodes and shuts them down[10].

2.3 Dynamic Voltage and Frequency Scaling

2.3.1 Idea

For the following discussion, we assume that a processor means a single core processor. The functioning of DVFS in a CMP is described in the next chapter.

The total power consumed by a processor (or generally, by any CMOS circuit) is approximately given by (Mudge [29]),

$$P \approx CV^2F.$$

where P is the power consumed by the processor, C is the total capacitance, V is the voltage supplied to the processor and F is the frequency at which the processor is running.
Since frequency is proportional to the voltage supplied, $P \propto F^3$.
The general formula is given by, $P \propto F^\gamma$, where γ is usually at least 2 indicating a non-linear relationship. Finally, energy is just the instantaneous power integrated over time.

The idea of DVFS is evident in this formula. Intuitively, if a processor runs at frequency F and consumes power P, reducing the frequency by a factor of X (i.e. new frequency is F/X) would result in a reduced power consumption of P/X^γ. In practice, however, such savings are difficult to achieve. As the frequency is decreased, the duration of any job running on the processor increases proportionally. As a result, reduction in instantaneous power does not necessarily mean a reduction in energy. Jobs may or may not benefit from DVFS depending

upon their profile (CPU bound or Memory/IO bound).

One also has to consider the energy consumption of the other server components such as the disk and memory, as well as that of the node itself during the extended duration when a job is run at lower frequency. When considering these issues, it might even be beneficial to run a job at a higher frequency so that it finishes early.

2.3.2 Literature

Many theory papers have investigated the DVFS model. Yao et al. [37] wrote a seminal paper on scheduling a set of jobs (where each job has an arrival time and a deadline, and requests for some number of CPU cycles) on a single variable-speed processor. They give an optimal offline greedy algorithm to create a minimum energy schedule and also propose two simple heuristics (namely, Average Rate Heuristic and Optimal Available) for online scheduling. Albers et al. [13] investigate the problem of scheduling a set of jobs on multiple variable-speed processors to minimize the total energy consumption of the schedule. They give a polynomial time algorithm for the case when the jobs have "agreeable deadlines", which simply means that a job that has an earlier release (submission) time compared to some other job should also have an earlier deadline compared to the other job. For the case of arbitrary release times and deadlines, they show that the problem is NP-Hard. A survey of the theoretical research on problems related to power management is given in Irani et al. [25]. Among others, it mentions the papers on the DVFS as well as those that discuss power-down strategies.

Freeh et al. [20] apply DVFS technique on a variety of HPC applications belonging to the NAS [5] and SPEC [6] benchmark suites (serial as well as parallel). They examine the effect of scaling on the performance and the energy consumed per application. Their experiments indicate that scaling the CPU to a lower frequency is beneficial only if the application is less CPU-bound. For example, for the NAS (cg) benchmark (that is less CPU-bound) on a single node, they report energy savings of 20% with a reduction in performance by only

3%. If a program is CPU-bound, they report that running it at the maximum frequency is beneficial for the performance of the program as well as for saving energy.

Elnozahy et al. [16] studied five different policies involving DVFS and Node Shutdown (they call it VOVO which stands for Vary On/Vary Off). They use data collected from Web servers in their simulations to evaluate each of the policies. Their results show that a combination of VOVO along with co-ordinated DVFS among nodes yields maximum energy savings (about 33 to 50%).

2.4 Miscellaneous

Kerstens et al. [28] put forth many ideas related to Green Computing using Moab. Using Wake-on-LAN or IPMI commands, Moab could intelligently shutdown or restart nodes as required depending upon the load on the cluster. Moab's scheduling policies could make use of data obtained using temperature sensors to schedule the workload on cooler nodes, as research indicates that the chance of node or component failure increases on nodes that run hot for prolonged periods. In addition to power-aware scheduling, they also look at cost-aware scheduling to take advantage of different energy costs (if any) during a day. Although the paper provides many useful ideas, no experiments on real clusters are mentioned; only a hypothetical example is given showing energy and cost savings for a company having huge size clusters.

Recent years have seen a rise in Cloud Computing Clusters (e.g. Amazon EC2). Garg et al. [21] consider a cloud made up of several heterogeneous data centers and investigate the scheduling policies for the cloud to reduce its overall energy consumption.

Scheduling algorithms and policies related to HPC clusters are important for reducing the energy consumption of a cluster. We defer the discussion of the related work on scheduling (classic scheduling theory as well as practical schedulers in HPC clusters) until Chapter 4.

2.5 Concluding remarks

Most papers [31, 15, 16] dealing with node shutdown have considered web-based traffic as their job model which is very different from the model of HPC clusters. Jobs in web clusters have a duration of seconds or a few minutes whereas jobs in the HPC clusters run for hours or days. Consequently, their analysis and results are not directly applicable to our cause. Our research is based on the ideas of Freeh et al. [19, 33, 34], Pinheiro et al. [31, 30] and Kerstens et al. [28], but we use an energy-aware scheduling algorithm for the complete cluster that incorporates both CPU Packing as well as Node Shutdown.

The existing research points to the fact that there are no common solutions that work for all clusters. Any energy saving technique in a cluster should consider the job characteristics, the job distribution within a cluster and the model of the cluster.

Chapter 3

Initial Investigations

In this chapter, we investigate the power management features currently available in modern servers and experiment with them on a test node. The observations and results of the experiments form the basis of the energy measurements used in our simulations. At the end, we investigate Moab's Green Computing feature.

3.1 Test framework

In order to explore the power management features, we need (i) a test node, (ii) jobs with different characteristics, and (iii) a power measuring device.

Our test node is a custom built server consisting of two Intel Xeon E5205 Wolfdale (dual-core) processors, 2 GB memory and 40 GB disk. Each processor has an L2 cache of 6MB that is shared by its cores. Additionally, each processor also has EIST (Enhanced Intel Speedstep Technology) capability allowing it to run at different frequencies; actually only two frequencies (1.6 GHz or 1.86 GHz) are supported by this processor. The operating system on the test node is Linux similar to the nodes belonging to the HPC cluster.

We use jobs from the SPEC CPU2000 benchmark suite [6]. It consists of various programs that have similar characteristics to the jobs that run on the cluster. All the programs are CPU intensive with the CPU utilization close to 100%, but there is a lot of variation in their memory profiles which causes different power consumptions.

We have used Watts Up Pro! [7] to measure the power consumption of the test node.

3.1.1 Basic Experiments

The following programs from the SPEC CPU2000 have been used for experimentation:

1. Vortex (Object-oriented Database) - Its memory requirement is in the range 38-60 MB. During a single run, the memory profile increases from 38 MB to 60 MB a couple of times.
2. Mcf (Combinatorial Optimization) - Its memory requirement was close to 75 MB during the tests (although SPEC mentions its maximum memory usage as approximately 190 MB). It linearly increases from 10 to 75 MB during the run.
3. GAP (Group Theory Computation) - Its memory requirement was approximately 200 MB.
4. Crafty (Chess playing): It has a very small memory requirement (less then 3 MB). All the memory used by the program could easily fit into the cache.

Note that for Vortex, Mcf and GAP, the memory used by those programs would not fit into the cache.

Table 3.1: Power Consumption & Runtime of a single instance of a SPEC Program

SPEC Program	Avg. Node Power (in watts)	Runtime (in minutes:secs)
None (Idle node)	119 W	-
Crafty	133 W	28:04 mins
Vortex	134 W	25:00 mins
Mcf	137 W	11:35 mins
GAP	138.5 W	15:28 mins

Using the default SPEC config file for Linux, we ran the programs in isolation on the test node for 12 iterations. We measured the power consumption of the node and the runtimes of the programs (the runtimes are used later to compare the effect of CPU Packing on the performance of programs). Based on the power consumption of Crafty (Table 3.1), we can deduce that the CMP is the biggest consumer of energy (about 15 W) in a node even with only one of its cores being busy. We can also see that for programs having a higher memory usage (Mcf or GAP), the increase in power consumption relative to a program that uses very little memory (Crafty) is approximately 4-5 Watts. Although we do not know

how to measure the power consumption of memory within a node, this seems to be a good approximation of the increase in power consumption of memory in a node while running a job that has medium to high memory usage.

Table 3.2: Power consumption of node with different number of busy cores

State of the node	Avg. Power Consumption (in watts)
Node when booted up but idle	119 W
One core busy	138.5 W
Two cores busy	155 W
Three cores busy	163 W
All four cores busy	170.5 W

To measure the power consumption of the node from no load to full load, we varied the number of instances of the GAP program running on the node from one to four. Table 3.2 shows the power consumption of the node corresponding to the number of instances of GAP running on it. By default, Linux balances the load among the CMPs. As a result, the first two instances run on idle CMPs and consume about 17-19 Watts, whereas the latter two instances run on already busy CMPs causing a relatively smaller increase of approximately 8 Watts in the power consumption.

3.2 Power Management Features

Power management features were first introduced in laptops and mobile devices due to their limited power capacities. Later, it was realized that even servers could benefit from some of those features. In the following subsections, we look at the hardware and software support for these features in modern servers.

3.2.1 CPU Packing

Siddha et al. [33] describe the CPU Packing feature in Linux. The file "*/sys/devices /system/cpu/sched_mc_power_savings*" is used to turn on/off the feature. It can be set using the command,
echo 1 > */sys/devices/system/cpu/sched_mc_power_savings*.
The only permissible values that could be written to this file are 0 and 1 under Linux 2.6.22.5 kernel. Future releases might support more values even in the user mode.

Linux does not strictly enforce packing of processes when CPU Packing is turned on. If the load on one CMP is high, Linux could migrate one or more processes to another CMP as required and again move it back after some time, as can be seen using the command *"mpstat -P ALL 5"*.

If a user wishes to enforce strict packing, the processor affinity feature could be used. The *taskset* command allows the user to specify the core(s) on which the job should be run using a bitmask. Each bit in the bitmask represents a core in the node. For example,
taskset -p [bitmask] [pid], where *pid* is the process id.

3.2.1.1 Experiments

Packing uses less power, but may not reduce energy. We have observed that when two jobs are packed on a CMP, together they consume less instantaneous power than if they were not packed i.e. run on different CMPs. An example of this was given in Chapter 1. Whether or not they consume less energy when packed depends upon the job characteristics.

We started two instances of SPEC programs simultaneously on the test node with CPU Packing turned on. We then measured the power consumption of the node and the runtimes of the jobs as given in Table 3.3.

Comparing tables, Table 3.1 and Table 3.3, we can see that the performance of Crafty, Vortex and GAP doesn't decrease by much when CPU Packing is turned on. The performance of Mcf, however, decreases approximately by a factor of two. Although Mcf and

Table 3.3: CPU Packing vs. Load Balancing of SPEC Programs

SPEC Program	When packing is ON		When packing is OFF	
	Avg. Power (in watts)	Runtime(in minutes:secs)	Avg. Power (in watts)	Runtime(in minutes:secs)
Two instances of Crafty	140.5	28:10	145.5	28:05
Two instances of Vortex	141.5	25:54	148	25:01
Two instances of Mcf	144	20:17	153	11:34
Two instances of GAP	148.5	16:10	155	15:34
Vortex and Mcf	141	26:43(Vortex) and 12:30(Mcf)	149.5	24:56(Vortex) and 11:38(Mcf)
Mcf and GAP	145.5	14:16(Mcf) and 16:48(GAP)	154	11:28(Mcf) and 15:29(GAP)

GAP both have a relatively high memory usage (in fact GAP uses more memory than Mcf), Mcf's performance decreases a lot whereas GAP's performance doesn't. We think that the reduction in performance of a job due to CPU Packing depends only upon the job's cache locality. It seems that the GAP program has a good cache locality whereas Mcf doesn't. Even if a process has higher memory usage, if the CPU is able to get the data from the cache most of the time, then it behaves almost like a CPU bound process. To summarize, we think that for CPU-bound processes, CPU Packing is surely beneficial. For memory/IO bound processes, it seems that only if the process has a good cache locality, CPU Packing would be beneficial. It also seems that if a job having a poor cache locality is packed along with a job having a good cache locality or one that uses less memory, the performance of both the jobs decreases only slightly.

In our simulations, we assume that the Packing has *no* effect on the performance of the jobs. The main reason for this assumption is because of the fact that in the HPC

cluster, Moab runs multiple jobs on a node alongside each other and the runtimes of the jobs as recorded in the logs (used for our simulations) already account for any change in their performance. Also, there is no way for us to tell how a job's performance would have been affected due to CPU Packing.

3.2.2 Advanced Configuration and Power Interface (ACPI)

ACPI [1] is a specification that outlines a standard interface that an operating system could use to configure and manage power related features provided by computer systems. Most current hardware vendors adhere to this specification. It describes different types of power states:

i. **Sleep states (S0 to S5)** : S0 is the working state of a node, S1 is Standby (in Linux), S4 is Suspend to Disk (Hibernate in Windows) and S5 is the shutdown state.

ii. **Processor states (C0 to C3)** : C0 is the working state of a processor. In any other C state, the processor is not active.

iii. **Performance states or P-states (P0 to PN)** : If the processor state is C0 (the processor is doing some work), P0 state indicates the highest frequency at which the processor is capable of working. As the processor successively moves to higher P states, it works at lower frequencies and thereby consumes less power.

As summarized in Table 3.4, experiments with our test node show that the S4 and S5 sleep states provide the maximum power savings. The price to pay when putting a node in these states is the relatively larger transition time to the S0 state when new jobs have to be run on the node.

ACPI Sleep State	Power Consumption (in watts)	Node Availability Time
S0	119 W	0 seconds
S1	95 W	$<$ 10 seconds
S4, S5	4 W	2-3 minutes

Table 3.4: Test node power consumption (in watts) vs. Transition time to S0 state

3.2.3 Dynamic Voltage and Frequency Scaling (DVFS)

Most modern processors implement the performance states (P-states) described in the ACPI specification; for example, Intel Enhanced Speedstep and AMD PowerNow!. This technology is popularly known as DVFS. In a single core processor, DVFS is simple. Reducing the voltage supplied to the processor also reduces its frequency and collectively they reduce the power consumption of the processor. In case of CMPs, DVFS is a little more complicated due to dependencies between the cores (Siddha et al. [33]). All the cores of a CMP are designed to operate at the same voltage level. The frequency of the CMP depends upon whether any of its cores are busy or not. If none of the cores are busy, the frequency is set to the minimum possible level (highest P-state). If at least one core is busy, the CMP runs in one of the lower P-states and all the cores (even the idle ones) operate at the frequency corresponding to that P-state.

In all cases (single core as well as multicore), the processor can transition between P-states only when it is in C0 (working) state.

In Linux, the *"cpufreq"* module has to be installed to take advantage of DVFS. Once installed, it has to be configured to use "ondemand" governor; the governor operates the CMP at the maximum frequency whenever there is work to be done and transitions it to the minimum frequency when all the cores are idle.

We were not able to find any user-space applications that allow the user to control the CMP frequency. As a result, we could not measure the power consumptions of a CMP when it is operating at different frequencies. Also, the CMPs on our test node support only two frequencies which limits the scope for experimentation. Therefore, we have not considered DVFS in our simulations.

3.3 Energy and cost bounds for the cluster

The MSU HPC Center has two main clusters, the Intel cluster and the AMD cluster, each consisting of 128 nodes. We were able to obtain the power consumption readings only for a node in the Intel cluster, but we think that the readings should be similar for a node in the AMD cluster.

The base power consumption of a node in the Intel cluster is approximately 220 W and when it has maximum load, it is approximately 285 W. Therefore, for the whole Intel cluster with 128 such nodes, we can obtain the lower and upper bounds for the energy consumption of the cluster and the cost of operation. We assume that the cost of energy is 10 cents per kWh.

If all the nodes in the cluster are in the ON state but are idle all the time, the instantaneous power would be 220*128 = 28,160 W. For an hour, it would be 28.16 kWh. The cost of operating the cluster for one hour is approximately $2.816. For a day, it is $67.584 and for an year, it would be approximately $24,668. This gives us the lower bound.

On the other hand, if all the nodes in the cluster are on and working at full load all the time, we can calculate the upper bound for cost which comes out to be approximately $32,000 per year.

With two such clusters and taking into account the cost for cooling the clusters, the total energy cost per year of the two clusters would be in the range $100,000 to $120,000.

3.4 Moab Green Computing

The Moab scheduler [4] offers the Green Computing (GC) feature that provides the ability to shut down nodes when they are idle. We experiment with this feature on a small test cluster.

Test Cluster Setup: The cluster has a main node that hosts the Moab and Torque servers. It also has four compute nodes each having a single-core processor, 512 MB RAM and 40GB

disk. All the compute nodes have wake-on-lan configured so that they could be shutdown or booted up remotely.

The Green Computing feature can be configured as given in [9]. The important configuration parameters are:

1. MAXGREENSTANDBYPOOLSIZE - It configures the number of nodes that are guaranteed to be ON at any time instant. A fixed set of nodes belonging to the cluster are assigned to the "green pool" and are always ON even if they are idle. We refer to the nodes that do not belong to the green pool as the "non-green pool" nodes.
2. NODEIDLEPOWERTHRESHOLD - It configures the maximum time for which a node is allowed to idle before it is turned off.
3. NODEPOWERONDURATION - It configures the maximum time required to boot up a node.

The command "mdiag -G" can be used to view the real-time status of the nodes in the cluster. The nodes are reported as idle, busy or being powered on/off.

Additionally, Moab requires a site-specific script (program) to be specified in its configuration file that can automatically turn on or off the nodes in the cluster. While Moab makes all the decisions, it delegates the actual task of node shutdown/restart to this script.

3.4.1 Observations

Before we present our findings, we would like to state that our observations on the small cluster might not truly represent the behavior of Moab on a cluster of about hundred multi-CMP nodes. We are also not sure whether Moab has any built-in learning mechanism that allows it to make more intelligent decisions when it's run over a longer period of time.

1. Initially, the green pool was configured to contain two nodes, while the other two nodes were shutdown. When a job is submitted, Moab wakes up one of the shutdown nodes and runs the job on it once it is booted up. Although the two nodes belonging to the

green pool were idle, Moab doesn't use them (refer to the example given at the end of the section). As a second job is submitted, Moab again wakes up the other shutdown node and starts the job on this node once it is booted up. A third job submitted to the cluster is immediately started on a green pool node. As the jobs complete their execution, the nodes become idle. After the configured idle time, the non-green pool nodes are shut down by Moab.

2. If idle nodes are available in the non-green pool, jobs are scheduled on those nodes first. If and only if all the nodes in the non-green pool are busy and can not accomodate any new jobs, the nodes in the green pool are used by Moab.
3. If all the nodes in the cluster are up and busy, newly submitted jobs are scheduled randomly to run on any node in the cluster without preference to the green or non-green pools.
4. Moab seems to use a static green pool, i.e. a fixed subset of nodes in the cluster is considered to be a part of the green pool.

We shall see in the next chapter using simulations that even these simple policies used by Moab with Green Computing enabled could help in saving energy.

Example: Moab Green Computing in Action

```
vudayagi@surya:~/Test> mdiag -G
NOTE:  power management enabled for all nodes
  partition power-on duration:  180 seconds
  NodeID        State  Power    Watts    PWatts
  host-26559    Idle     On     200.00   200.00
  cottonmouth   Idle     On     200.00   200.00
  host-9553     Idle     On     0.00     200.00(powered off by Moab)
  host-26558    Idle     On     0.00     200.00(powered off by Moab)
vudayagi@surya:~/Test> qsub jobs.sh
1433.surya
```

```
vudayagi@surya:~/Test> mdiag -G
NOTE:  power management enabled for all nodes
  partition power-on duration:  180 seconds
  NodeID        State   Power   Watts    PWatts
  host-26559    Idle      On    200.00   200.00
  cottonmouth   Idle      On    200.00   200.00
  host-9553     Idle      On    0.00     200.00(powered off by Moab)
  host-26558    Busy      On    450.00   450.00
   NOTE:  node is currently being powered on
   (request power-2 started 3 seconds a go)
vudayagi@surya:~/Test>
```

Chapter 4

Theory, Simulations and Results

In this chapter, we begin with a brief discussion of the relevant scheduling theory. We then explain the ideas used in our scheduling algorithm that uses the energy estimates of running a job on the nodes of a cluster to make the scheduling decisions. After that, we describe the different simulations that we have performed in this thesis. Finally, we present our results and recommendations to the HPC administrators.

4.1 Cluster Scheduling Related Work

The HPC literature classifies all the scheduling algorithms as either time-sharing or space-sharing [24]. In the time-sharing algorithms, the resources are shared and no job might have exclusive access to them. Space-sharing techniques allocate the resources exclusively to a job until its completion. The Moab scheduler deployed in the HPC cluster uses the latter approach. The actual scheduling of jobs can be done using First Come First Serve (FCFS), Shortest Job First (SJF) or based on priorities (note that even FCFS or SJF can be viewed as scheduling based on priorities where jobs submitted earlier in time or short duration jobs respectively are given a higher priority). With any policy, the possibility arises that the job at the front of the queue may not be scheduled due to unavailability of resources. When this happens, it shouldn't cause the other jobs to wait in the queue although there are resources that could be used to start the job. To solve this problem, the scheduler can use reservations and backfill. Reservations are given to higher priority jobs that ensure that those jobs would not be starved by lower priority jobs and are guaranteed to run at some point of time in the future. Any lower priority job that does not violate the reservations could be started (backfilled) immediately.

There are two commonly used backfill algorithms: liberal and conservative. In liberal backfill, only the highest priority job waiting in the queue gets a reservation. If some other job (either newly submitted or one that's already waiting in the queue) doesn't violate this reservation, it could be backfilled. The opposite of that is conservative backfill in which all jobs waiting in the queue get a reservation. The algorithms themselves can be found in [18]. Srinivasan et al. [35] compare the two algorithms and show that conservative backfill reduces whereas EASY backfill improves backfilling opportunites. However, the **worst** case turnaround time of jobs is actually more when using EASY as compared to conservative approach due to some wide jobs (those that require more than 8 processors in their terminology) neither being able to get backfilled nor get a reservation. Moab documentation mentions that backfill improves the resource utilization within a cluster by about 20% and improves the turnaround time of the jobs even more [9].

A small survey of the available batch schedulers for clusters is given in [17]. Some of the popular ones being used in HPC clusters are Moab, Maui, LoadLeveler and PBS. The main algorithms used within Maui Scheduler (to do backfill, job prioritization, fairshare etc.) are described by Jackson et al. [26].

Note that the two main objectives of a scheduler in an HPC cluster are: to minimize the waiting time of jobs and to maximize resource utilization (equivalent to maximizing the cluster throughput). Using reservations and backfill, both objectives can be met.

For the sake of completeness, we briefly cover the classic scheduling theory relevant to our work. The HPC cluster consists of a large number of identical nodes and each node has multiple cores, some memory and disk. A job submitted to the cluster requests for one or more nodes, one or more cores per node, some memory and disk. In scheduling theory, this model is referred to as "Scheduling on Multiple Machines" or "Multiprocessor Scheduling", where all the machines or processors are identical.

Scheduling on multiple machines is a well researched problem. The basic aim is to schedule a job from a set of available jobs on one or more machines while trying to minimize

(or maximize) some criterion. Some of the commonly used criteria are the makespan (the completion time of the schedule), the mean waiting time, the maximum waiting time and the total flow time (flow time, also known as the turn around time, is the difference between job completion time and job submission time). Except for a few simple models, this problem is usually NP-Hard in the offline setting. When the scheduler does not have knowledge of the future jobs, it has to make the best possible scheduling decision for any job based on the current state of the cluster. The situation just described is an online scenario and it represents the way in which practical schedulers in HPC clusters work. Since our research is pertaining to HPC clusters, we are interested in online scheduling algorithms that try to minimize the average waiting time or the maximum waiting time of jobs on parallel machines. Using the notation, $\alpha|\beta|\gamma$, introduced by Lenstra et al. [27], our cluster could be classified as, $PN||W_{max}$, when the objective function is to minimize the maximum waiting time or as, $PN||\frac{1}{j}\sum_{j} W_j$, when the objective function is to minimize the average waiting time. Here, P indicates identical machines and N is the number of such machines available. The jobs are considered to be non-preemptive and without any precedence constraints. The only difference in our case is that all the jobs are not available at the start of scheduling (representing an online model).

Scheduling papers that consider waiting time as the objective function have been hard to find. Here, we look at some papers that deal with the flow time or the makespan. The performance of an online algorithm is usually compared with the performance of the optimal offline algorithm (for the same problem) and given in terms of the ratio of the former to the latter known as the competitive ratio. The most commonly cited online algorithm in literature is the Graham's list scheduling algorithm (online version) that attempts to minimize the makespan by scheduling a job (chosen from a set of jobs) as soon as a node becomes available in the cluster [23]. It has a competitive ratio of $2 - 1/m$. Shmoys et al. present many theoretical results related to online scheduling on parallel machines [32]. A classification of online scheduling problems for multiple processors is given in [14].

Finally, we note that even if the main objective using classic scheduling were to reduce the energy consumption of the nodes in a cluster, energy minimization cannot be used as the sole criterion. Otherwise, scheduling all the jobs one after another on a single node would always minimize the energy (although it would cause huge wait times for most jobs). Therefore, energy minimization has to be considered alongside a second objective function such as minimizing the makespan or total flow time.

4.2 Main idea for Energy-aware scheduling

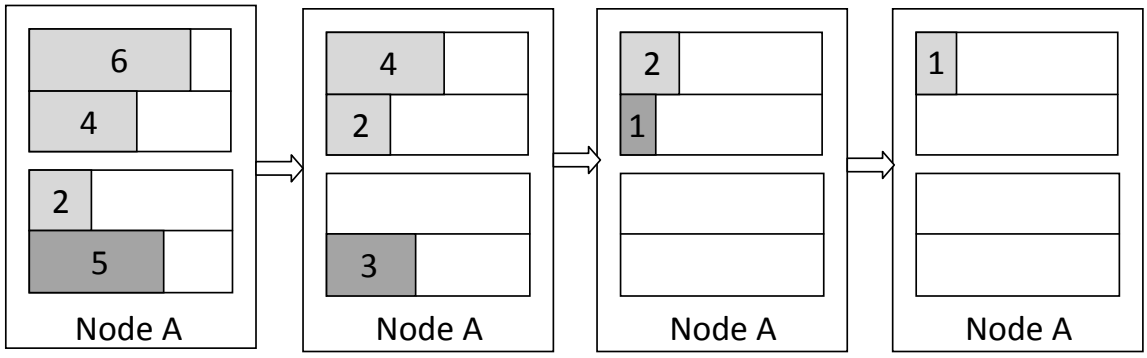

Figure 4.1: Job migration for CPU Packing within a node

In an SMP system, the operating system can migrate jobs across CMP boundaries within a node seamlessly in real time, although the jobs cannot be migrated from one node to another. This feature helps CPU Packing as jobs can be moved automatically between CMPs to save power. Figure 4.1 shows the migration process. The state of a node A running four jobs is shown along with the remaining processing time of each job. After two time units, one job completes execution causing one CMP to be half busy. After two more time units, another job completes execution thereby causing the other CMP to be half busy as well. At this instant, the operating system could pack the remaining two jobs on a single CMP to save power. It is assumed here that the performance of the jobs does not decrease due to the packing.

The migration of jobs using CPU Packing technique helps to save power (and possibly

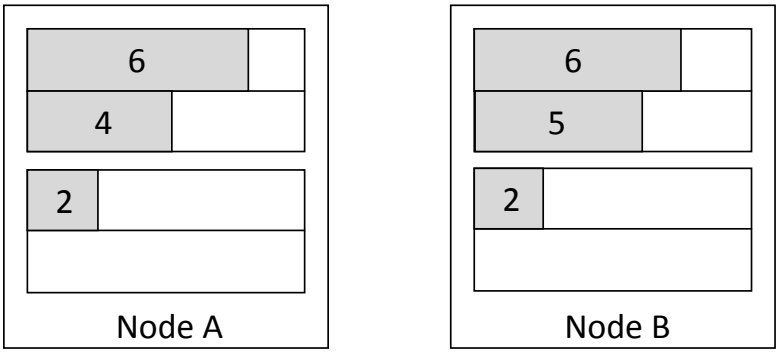

Figure 4.2: Snapshot of a cluster with two nodes

energy) within a node. However, a cluster consists of many nodes and to save energy at a cluster level, the scheduler has to decide which nodes would consume the least amount of energy to run a job. To see how this can be done, let us consider a cluster having two nodes (node A and node B) each having two CMPs with two cores per CMP. A snapshot of the cluster at some time instant is shown in Figure 4.2. Each node has exactly one idle core. On the busy cores, the remaining processing time (in minutes) of the jobs running on those cores is shown. If, at that instant, a job (let us call it job J) with a requirement of one core and 5 minutes processing time is submitted to the queue, the scheduler has to decide whether to start the job on node A or node B.

Table 4.1: Increase in power consumption (in watts) of node with different number of busy cores

State of the node	When packing is on	When packing is off
Idle node	0 W	0 W
One core busy	15 W	15 W
Two cores busy	21 W	30 W
Three cores busy	36 W	36 W
All four cores busy	42 W	42 W

The scheduler is aware of the processing time requirements of the jobs already running on the nodes. For the newly submitted job J, the scheduler can simulate starting the job

on node A (refer to Figure 4.1) and compute the energy consumption of the node (we have assumed that the operating system performs migration of jobs for CPU Packing on the node) until the job finishes on the node. It could also calculate the energy consumption of the node for the same duration if the job were not run on this node. The difference between the two energies is the energy consumed to run the job J on node A. Similarly, the scheduler could calculate the energy consumption of job J on node B as well. Whichever node consumes the least energy to run the job is chosen for scheduling.

Carrying on with our example, let the **increase** in the power consumption of a node (not the total power consumption of the node itself) as the number of busy cores on the node vary from zero to four, be as given in Table 4.1. Then, the energy consumption of node A during the time when job J runs on it (for 5 minutes) can be given by $(2 * 42 + 2 * 36 + 1 * 21) = 84 + 72 + 21 = 177$ Watt-mins.

If the job is not run on node A, it would still consume $(2*36+2*21+1*15) = 72+42+15 = 129$ Watt-mins, during the next five minutes.

The difference between the two energies $= 177-129$ Watt-mins $= 48$ Watt-mins is the energy consumed by job J on node A.

Similarly, if job J were to be started on node B, the energy consumed by the job on node B would be $(2 * 42 + 3 * 36) - (2 * 36 + 3 * 21) = 57$ Watt-mins.

Therefore, the scheduler should start the job on node A since it can do the same work for less energy. This is the main idea that we have used in our energy aware scheduling algorithm.

Important: Although we have assumed in this section that the nodes have CPU Packing turned on, the energy-aware scheduling approach remains unchanged even if the assumption is not made. Instead of using the "increase in energy" values for a node when packing is turned on, the scheduler can use the "increase in energy" values for the node when packing is turned off (refer Table 4.1) to make the scheduling decisions.

Before we conclude this section, it is important to understand that in the HPC cluster, a job could request multiple nodes and multiple cores per node (i.e. multiple tasks). In such

cases, determining which nodes collectively would consume the least energy to run the job is more complex. We cover this generalization later when we present our Packing Algorithm. In this section, we deliberately chose a simple example of a job requesting just one task (one core on a node) for the sake of clarity.

4.3 Introduction to Simulations and Setup

The HPC Cluster at MSU is used by various departments within MSU for their research. Before configuring any new policy for Moab (which is the scheduler being used in the cluster, as mentioned before), the impact of the policy on the users of the cluster has to be guaged. The Green Computing feature, in particular, involves shutting down nodes that could potentially cause larger wait times for jobs submitted by the users. For this reason, it is not possible to simply enable Moab's Green Computing feature on the main production cluster. The purpose of the simulations is to simulate the behavior of Moab with the Green Computing feature enabled on the cluster. Based on the results of the simulations, we could decide whether it is beneficial to enable the feature in the HPC cluster.

We perform three main simulations in this thesis:

1. First we perform the Moab simulation in which we recreate the state (nodes, jobs and queue) of the HPC cluster for a complete week when Moab had done the scheduling and generated the logs.
2. Next we do the packing simulation in which we begin with the same cluster state as Moab at some time instant, but use our energy-aware packing algorithm (described later) for doing the scheduling.
3. Finally, building up on the packing simulation mentioned above, we simulate node shutdown using Moab's green computing policy. This simulation is the closest we can get to running Moab with Green Computing on the HPC cluster.

For all the simulations, we calculate the energy consumption of the cluster during the

week. Ultimately, the main purpose of the simulations is to help us understand whether any energy savings are possible in the cluster and if the answer is affirmative, to quantify the savings and/or pain involved while using any of the energy saving techniques.

4.3.1 Log files

All the simulations use log files collected from the HPC cluster during a period of six months from 5th March 2008 to 15th August 2008. To alleviate any privacy concerns, the logs were anonymized. The log files contain log data from the Intel cluster as well as the AMD cluster and it is clear that for many weeks the usage of the Intel cluster was consistently higher than that of the AMD cluster (relatively fewer jobs were submitted to the AMD cluster during those weeks). Therefore, we have just used the Intel cluster logs for our research.

The log files contain different types of events, namely:

1. JOBSTART: It is logged when a job is started. It also contains details of the job such as the number of nodes, cores per node, memory, walltime etc. requested. A description of the various fields in an event can be found in Chapter 16 of the Moab Admin Guide [9].
2. JOBEND: It is exactly same as the corresponding JOBSTART event except that for this event, the job completion time field contains the actual completion time of the job.
3. JOBCANCEL: It is similar to a JOBEND event, but is logged when a job is canceled for some reason.
4. NODEUP/NODEDOWN: These events are logged when a node is considered available or not for scheduling purposes. These events do not necessarily indicate that a node was turned off or on.

Moab supports some other types of events like JOBSUBMIT, JOBHOLD, JOBREJECT, NODEFAILURE etc., but these are not available to us in the log file.

4.3.1.1 Problems with Logs

Ideally, for every job submitted to the cluster, there should be exactly one JOBSTART and JOBEND events (or a JOBCANCEL event) in the log file. Occassionally, we have noticed some entries that do not follow this rule and if not resolved, cause problems in the simulations. Below is the list of problems in the logs along with a discussion of how we have chosen to resolve them.

1. Some JOBSTART events do not have corresponding JOBEND events in the logs. In the Moab simulation, we allow such jobs to run until they reach their walltime limit or until there is a conflict with some other job that had run without any problems. In case of a conflict, it is assumed that the job's completion time is equal to the time of the conflict.
2. A very few JOBCANCEL events do not have corresponding JOBSTART events. These events also don't carry the information about the nodes on which they were running. Without the information about the nodes, it's not possible to assign these jobs to any nodes while recreating the state of the cluster during Moab simulation. We ignore such jobs.
3. On some rare occasions, there are multiple JOBCANCEL events for a job. The job completion time is also different in each of these entries. We resolve this issue by choosing the time of the chronologically last JOBCANCEL event as the job's completion time.
4. A few jobs have multiple JOBSTART events recorded in the logs. This might happen when the nodes on which the job was running were shutdown for some reason and the job had to be requeued and started again at a later time. This problem is similar to some jobs not having JOBEND events and has been resolved in the same fashion.

We had earlier thought of ignoring the problematic jobs in our simulations, but on further investigation, we realized that some of those jobs had huge resource requirements and ignoring them would not give the best possible estimate of the overall energy consumption of the cluster. Therefore, we incorporated the problematic jobs in the Moab simulation

and calculated the estimates of their runtimes. Those runtimes are then used in the other simulations. Since the jobs used by all the simulations are exactly the same (same resource requirements, submit times, runtimes etc.), there is no unfair advantage either to Moab or to our simulations, thereby giving us unbiased results.

4.3.2 Policies used by Moab on the live cluster

Moab supports a lot of different policies, but only a few of those are important for scheduling of jobs. We have tried to make our simulations as much similar to Moab as possible and in that process, we have implemented the following policies:

1. *Priority of jobs* - Moab calculates the priority of a job based on a number of factors as described in Chapter 5 of the Moab Admin Guide [9]. There are seven main components within Moab that contribute to the priority of a job. These components further contain a number of sub-components. Each of these components and sub-components are assigned weights in the Moab configuration file. The formula used to calculate the priority of a job is:

   $\sum$ (Component Weight) * ($\sum$ (Sub-component Weight * Sub-component Value))

   In the MSU cluster, only two of the components (SERV and RES) are used and the weights assigned to them in the config file are:

   ```
   RESWEIGHT = 5

   PROCWEIGHT = 10000

   MEMWEIGHT = 2

   SERVICEWEIGHT = 1

   QTIMEWEIGHT = 1
   ```

   If C and M are the number of cores requested and memory requested by a job respectively, and Q is the time (in minutes) spent by the job waiting in the queue for service, the priority of a job in the HPC cluster is given by,

   ```
   RESWEIGHT*(C*PROCWEIGHT + M*MEMWEIGHT) + SERVICEWEIGHT*(Q*QTIMEWEIGHT).
   ```

Substituting the weights with their configured values, the formula being used to calculate the priority is, `5*(C*10000+M*2) + 1*(Q*1)`.

The weighting values and the priority of a job can be verified from the output of "mdiag -p" command, as shown below.

```
---------------------------------------------------------
Job Id PRIORITY Cred(Class) Serv(QTime) Res( Proc:  Mem)
Weights           1( 1)       1( 1)       5(10000:  2)
5998793 1693529   0.0(0.0)   0.7(11609)   99.3(32.0:8192.)
5998794 1643449   0.0(0.0)   0.7(11609)   99.3(31.0:8184.)
---------------------------------------------------------
```

Intuitively, these weights cause the jobs that request a large number of cores to have higher priorities. The scheduler attempts to start those jobs first while the jobs with small resource requirements are used to fill the gaps in the cluster using backfilling.

In our simulations, we ignore the queue time used to calculate the priority of a job for two reasons. First, its contribution to the priority value is very small (mostly $< 1\%$). Second, Moab documentation mentions that the queue time of a job is not necessarily the time spent waiting in the queue since the jobs submission, rather it is calculated based on certain fairshare policies that we do not take into account in our simulations.

2. *Liberal (EASY) Backfill* - In the cluster config file, the RESERVATIONDEPTH parameter has been set to "1" which indicates that Moab uses liberal backfill. As mentioned earlier, in liberal backfill, only the highest priority job waiting in the queue gets a reservation and any job that does not violate the reservation could be started immediately in the cluster. The algorithm used for backfill in the cluster is FIRSTFIT.
3. *NODEALLOCATIONPOLICY* - When scheduling a job, this policy determines which node(s) should be allocated to the job. In the cluster config file, this policy has been set to PRIORITY and the priority function used to calculate the node priority is, `(PRIORITYF=-1.00*APROCS-1.00*CPROCS)`, where `APROCS` is the number of available

cores on the node and `CPROCS` is the number of configured cores on the node. Since `CPROCS` is constant for all the nodes, this priority function just dictates that a job should be started on the nodes having the least number of available cores.

4. *JOBNODEMATCHPOLICY* - This policy adds more restrictions on how the nodes are selected for scheduling. If this policy is set to EXACTNODE, Moab needs to allocate the exact number of distinct nodes requested by the job. If it is set to EXACTPROC, Moab needs to allocate nodes that have exactly the same number of idle cores as requested by the job.

   In the cluster config file, the *JOBNODEMATCHPOLICY* is neither EXACTPROC nor EXACTNODE. The Moab documentation doesn't explain the behavior of Moab when this policy is set to default. Fortunately, this information can be extracted from the log file. It can be seen that for a job requesting some number of nodes and cores per node (say, X nodes and C cores per node), Moab doesn't necessarily allocate X distinct nodes to the job. Instead, it tries to allocate multiple tasks on a node whenever possible. For example, if a job requests 3 nodes and 1 core per node, Moab might allocate a single node with 3 idle cores to the job. We have implemented this behavior in our simulations.

4.3.3 Assumptions in simulations

1. We assume that all the jobs running in the cluster have a CPU utilization of 95-100%. This assumption is not unrealistic as can be verified on the compute nodes of the HPC cluster that most cores running user jobs have close to 95% CPU utilization. Even Ganglia [2] reports similar load statistics for the cluster.
2. We assume that CPU Packing of jobs does not decrease their performance. In other words, shared resources are not a bottleneck for any job submitted to the cluster.
3. The log event does not contain information about the disk usage of the jobs. Therefore, we consider only the number of nodes, cores per node and the memory requested to make scheduling decisions.

4. The number of nodes that were available to Moab for scheduling cannot be determined accurately from the logs. However, in order to do a fair comparison between our simulations and Moab, at any instant, our simulator should not use more nodes than those that were used by or at least available to Moab. Otherwise, we have noticed in our simulations that due to the availability of more nodes, the simulator is able to run more jobs in the cluster at times when Moab was unable to do so.

   We tackle this problem as follows. First, the maximum node usage by Moab at any time during a week is determined while running the Moab Simulation. This value gives us a good approximation of the maximum number of nodes available to the other simulations during that week. Second, we divide the complete week into intervals of three hours and find the maximum node usage by Moab (in the Moab Simulation) during these intervals. During the other simulations, the scheduler is not allowed to use more nodes than the maximum number of nodes used by Moab during that interval. The rationale behind these assumptions is to give Moab the advantage in terms of the number of nodes that were available for scheduling. The other simulations then simply make use of that information while scheduling.

4.3.4 Simulation parameters

1. The time required to boot up a node is assumed to be 100 seconds based on our experiments with the test server. Although the test server appears to be up within a minute, it still performs various startup activities during the next minute (the power consumption of the server stabilizes after that).
2. The time required to shutdown a node is assumed to be a minute.
3. The power consumption of a node in the cluster (booted up but idle) is 220 Watts. When the node is fully busy running jobs, the power consumption is 285 Watts. These readings were obtained from a node in the Intel cluster using the Watts-Up Pro measuring device.
4. During the startup and shutdown of a node, the power consumption is assumed to be

245 Watts on average. This reading was again observed on a node in the Intel cluster using the Watts-Up Pro measuring device.

5. When a node is shutdown, it is assumed that it still consumes 20 Watts. The reason for this is that each node has a Baseboard Management Controller (BMC) Chip that remains active to facilitate remote administration of the node even when a node is shut down.
6. If a node has been idle for 5 minutes, it is shutdown by the scheduler.

4.3.5 Simulation Steps

For all the simulations, we just use the JOBEND event, as this event contains all the information related to a job i.e. submission time, start time, completion time, number of nodes requested, number of cores per node requested, memory requested, walltime requested etc. For the problematic logs, the job end time is calculated as described in the "Problems with Logs" section. From a high level, the main steps involved in a simulation are:

1. Choose a simulation start date. The simulation start time is always at 12 AM on the start date. The simulation end time is always a week in future from the simulation start time.
2. The job queue is built using the JOBEND events from the logs:

   a) If a job's submit time is less than the simulation start time and its start time is greater than the simulation start time, we add the job to the wait queue.

   b) If a job's start time is less than the simulation start time and its end time is greater than the simulation start time, we assign the job to the nodes on which it ran. This is the case when the job had already started running on the nodes. c) If a job's submit time is greater than the simulation start time but less than the simulation end time, we put the job into the wait queue.
3. The scheduler considers the job at the front of the queue (having the highest priority) for scheduling.

The last step includes the scheduling algorithm and it is different for each simulation. The details of the algorithms are given in the following sections.

4.4 Moab Simulation

The main purpose of the Moab simulation is to compute results that could be used as a baseline for comparison with the other simulations. In the Moab simulation, the energy consumption of the cluster, when using Moab without the Green Computing feature enabled, during a week is calculated.

The Moab simulation simply schedules the jobs in the queue based on their actual start times from the logs. In fact, there is no scheduling involved; just the assignment of jobs onto the nodes in the cluster on which they ran. The results for this simulation are presented in the later sections when it is compared with the other simulations.

4.5 Packing Simulation

The algorithm used in the packing simulation uses two important ideas with respect to packing jobs. First, it tries to pack the nodes in the cluster as much as possible with jobs (node-level packing). This approach is similar to Moab's node allocation policy of giving more priority to nodes that have fewer idle cores when scheduling a job. Second, it uses an energy-aware scheduling approach that assumes that CPU Packing is enabled on all the nodes of the cluster. Details of the algorithm are given below:

Packing Algorithm:

Consider a job that requests S tasks where the definition of a "task" is a logical unit requiring one node, c cores per node, m memory per node and a walltime W as described in Chapter 1.

1) Set $k = c$ and $num\_tasks = 0$.

2) Check all nodes in the cluster that currently have exactly k idle cores and have at least

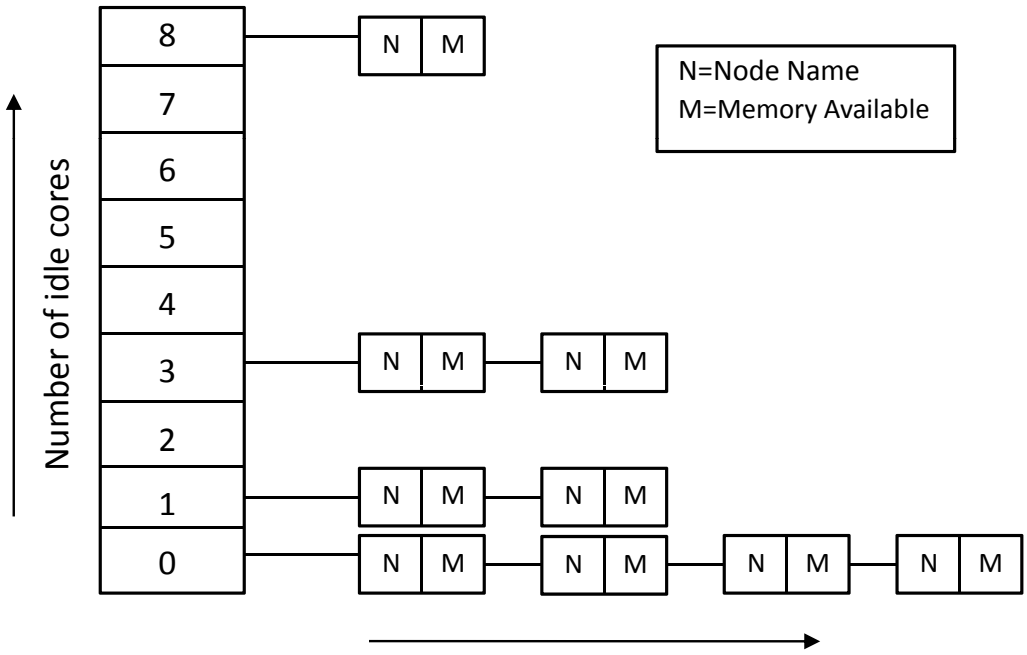

Figure 4.3: Data Structure used for Packing

m memory available (Refer Figure 4.3).

a) If those nodes are capable of running exactly (S - num_tasks) tasks, allocate the nodes to the job. Return.

b) If those nodes are capable of running more than (S - num_tasks) tasks, find the nodes for which the job's energy consumption would be minimum (details given below).

c) Else, commit the nodes selected in 2(a) to the job. Set num_tasks to the total number of tasks that could be run on the committed nodes. Set $k = c + 1$. Goto step 2.

3) If resource requirements are not met, put the job in the waiting queue.

Step 2(a) ensures that nodes are packed to their maximum whenever possible.

In step 2(b), the scheduler needs to find the nodes that would consume the least energy for the job. To solve this problem, we need a generalization of the scheduling algorithm that we had presented in section 4.2. The problem can stated formally as:

There are N nodes labeled as R_j ($1 \leq j \leq N$) and P identical tasks such that each task requires a processing time of W. Each node has the same number of total cores (Z) as well as idle cores. The node R_j can run a maximum number of tasks given by U_j, where

$0 \leq U_j \leq Z$. For any two nodes, R_j and R_k where $j \neq k$, U_j might be different than U_k depending upon the memory available on the nodes and the memory requested by the job. The number of tasks, $P \leq \sum U_j$ $(1 \leq j \leq N)$, indicates that it is always possible to run all the P tasks on a subset of the nodes, thereby guaranteeing a solution. Let $F(j,k)$ $(0 \leq k \leq U_j)$ be the energy consumption of the node R_j if k tasks are started on the node at the current time and they run for the duration W. Let $E(j,k) = F(j,k) - F(j,0)$ be the **increase** in the energy of node R_j if k tasks are started on the node at the current time and they run for the duration, W. Refer to the scheduling method in Section 4.2 to see an example of how $E(j,k)$ values could be computed.

Given the above definitions, the main objective is to,

$$\text{minimize} \sum_{i=1}^{N} \sum_{j=0}^{U_i} E(i,j)x_{ij}$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j=0}^{U_i} jx_{ij} = P, \qquad (1)$$

$$\sum_{j=0}^{U_i} x_{ij} = 1, \text{ where } 1 \leq i \leq N \qquad (2)$$

$$x_{ij} \in \{0,1\} \text{ , where } 1 \leq i \leq N \text{ and } 1 \leq j \leq U_i \qquad (3)$$

The second constraint imposes the restriction that a node cannot be selected more than once in a solution.

As we see below, a polynomial time dynamic programming solution exists for this problem.

Dynamic Programming formulation:

Let $T(M,Q)$ be the minimum energy required to run Q tasks on M nodes, where $1 \leq M \leq N$ and $0 \leq Q \leq P$. Our main aim is to find the minimum energy required to run P tasks on N nodes given by $T(N,P)$.

The recurrence relation for $T(M,Q)$ can be written as:

$$T(M,Q) = \min_{0 \leq j \leq min(Q,P)} T(M-1, Q-j) + E(M,j),$$

where,

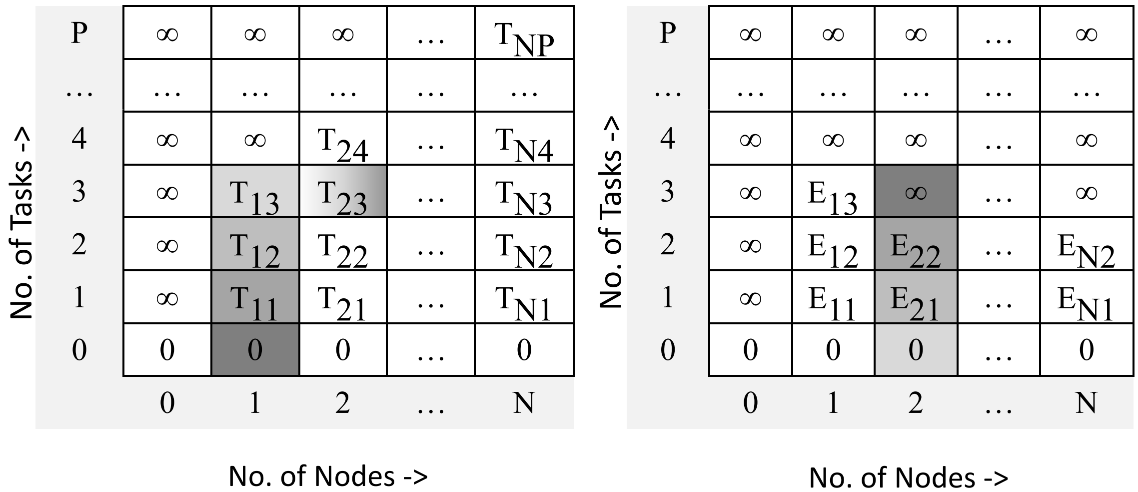

Figure 4.4: Dynamic Programming Solution

$T(0, Q) = \infty$, where $0 < Q \leq P$,

$T(M, 0) = 0$, where $1 \leq M \leq N$,

$E(0, Q) = \infty$, where $0 < Q \leq P$,

$E(i, j) = \infty$, where $1 \leq i \leq N$, $j > U_i$

$E(M, 0) = 0$, where $1 \leq M \leq N$.

The idea behind the recurrence is as follows. We know that the M^{th} node can run a maximum of U_M tasks. For $0 \leq j \leq U_M$, if the M^{th} node runs j tasks out of the total Q tasks, the energy consumption of those j tasks on M^{th} node is given by $E(M, j)$. The remaining $Q - j$ tasks now have to be assigned to the remaining $M - 1$ nodes such that the energy required to run those tasks is minimized. Thus, we have an exact subproblem $(T(M-1, Q-j))$ of the original problem. Finally, the minimization function in the recurrence chooses the j for which the sum of the minimum energy required to run $(Q - j)$ tasks on $M - 1$ nodes and the energy required to run j tasks on the M^{th} node is minimized.

This recurrence could be computed in polynomial time by maintaining a table that stores the results of the sub-problems. Figure 4.4 shows the tables for $T(N, P)$ and $E(N, P)$. The values in the $E(N, P)$ table are known. Using the $E(N, P)$ table, the values in the table $T(N, P)$ are computed in a bottom up and left-to-right fashion.

Note that the table for $T(N, P)$ only gives the minimum energy required to run P tasks on N nodes. It doesn't give the actual nodes to be used and the number of tasks to be run on those nodes. To get that information, another book-keeping table has to be maintained.

We use this energy-aware scheduling algorithm, that assumes CPU Packing on individual nodes, in our "Packing Simulation".

4.5.1 Comparison between Moab and Packing

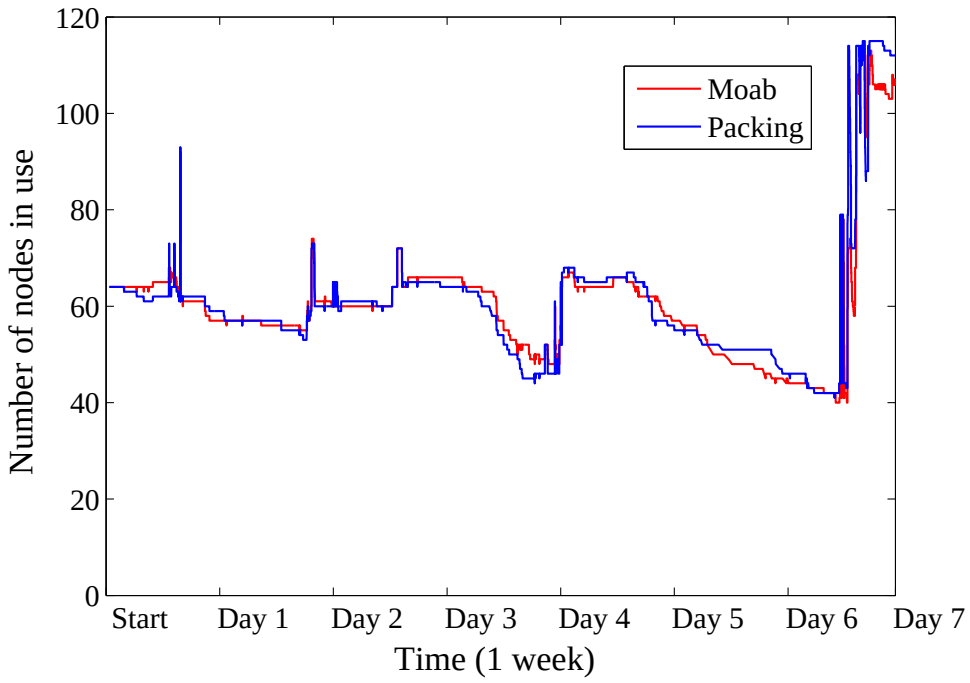

Figure 4.5: Moab vs. Packing: Node usage during a week.
"For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this thesis."

Figures 4.5 and 4.6 show the nodes and cores usage in the cluster for the Moab and Packing simulations. The Moab simulation is just a recreation of the scheduling done by Moab in the HPC cluster, whereas Packing simulation uses the packing algorithm mentioned above. As seen in the graphs, the scheduling done by the packing algorithm is very similar to the scheduling done by Moab giving us the confidence that any results obtained using the

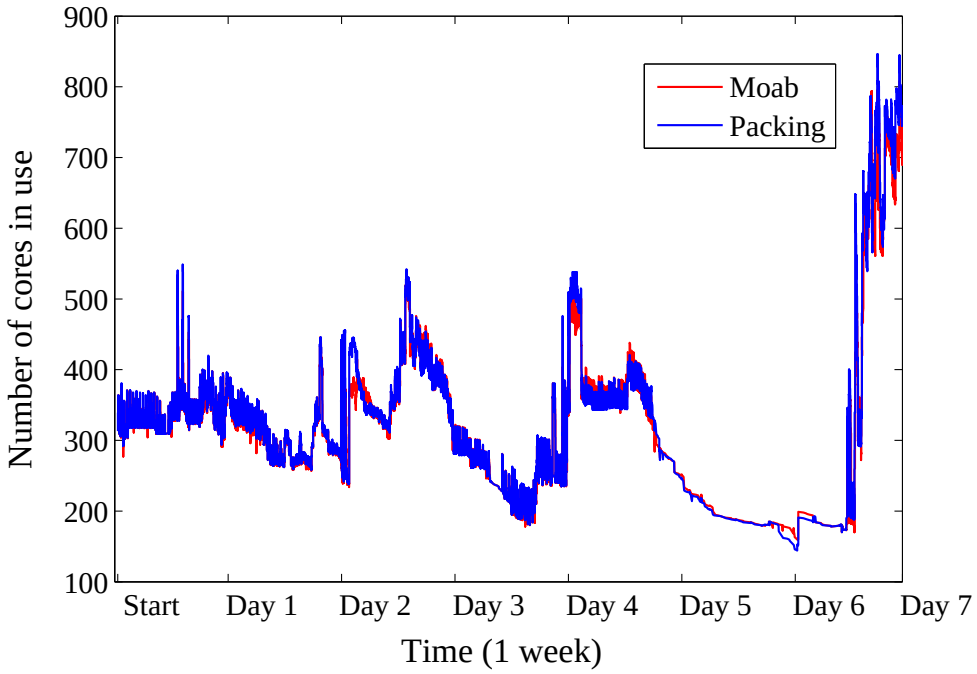

Figure 4.6: Moab vs. Packing: Core usage during a week

packing algorithm are comparable with Moab. Occasional spikes seen in the graphs were caused due to many jobs getting submitted to the cluster at once and running for very short durations. They occur in Moab scheduling as well as in the Packing simulation.

Table 4.2 shows the energy consumption of the cluster when using Moab and the Packing Algorithm respectively. For all the weeks in the table, the cluster consumes less energy when using the Packing algorithm as compared to Moab but by a very small amount (less than 1%). It surprisingly shows that the energy-aware scheduling approach used for the simulation would have little impact on the energy consumption of the cluster. Although Moab uses a different scheduling algorithm that does not consider the resulting energy consumption of nodes when scheduling jobs (as far as we know), the total energy consumption of the cluster with either of the scheduling algorithms seems to be almost the same.

Table 4.2: Energy consumption: Moab vs. Packing Simulation

Week start date	Using Moab (kWh)	Using Packing (kWh)	%Energy Savings
1st April	4,735.47	4,707.38	0.59
5th April	5,068.34	5,038.57	0.59
10th April	5,176.74	5,137.19	0.76
15th April	4,514.14	4,478.52	0.79
22nd April	4,616.89	4,586.38	0.66
29th April	4,728.40	4,703.74	0.52
5th May	3,980.82	3,948.06	0.82
12th May	4,795.84	4,776.66	0.4
15th May	4,811.33	4,786.83	0.51
22nd May	4,655.44	4,633.60	0.47
1st June	4,689.71	4,673.45	0.35
7th June	4,434.60	4,425.78	0.2
13th June	4,337.75	4,298.96	0.89
1st July	4,975.17	4,964.62	0.21
15th July	4,562.90	4,543.82	0.42
18th July	4,585.59	4,551.12	0.75
25th July	5,466.45	5,460.17	0.11
1st August	5,552.98	5,537.79	0.27

4.6 Moab Green Computing Simulation

4.6.1 Algorithms

In Chapter 3, we presented our observations related to Moab Green Computing that uses the node shutdown concept to reduce energy. We use those observations in our simulation in the form of the scheduling algorithm given below. Note that, for all the algorithms given below, although not explicitly mentioned, a node in the cluster is shutdown by the scheduler if it has been idle for more than five minutes.

Algorithm:

Moab GC (Green Computing):

For a job that requests N tasks,

1. Determine whether the job could be started on the active nodes in the cluster **not** belonging to the green pool. If yes, start the job.
2. If not, check the nodes that are currently shutdown (if any) and wake them up if the job could be started using them. Start the job when the nodes have been booted up.
3. If there are no nodes in the shutdown state, check if the job could be started using the green pool nodes.
4. If the job cannot be started after all these considerations, put the job in the waiting queue.

The problem with the Moab Green Computing approach is that for a job, if non-green pool nodes are not available, the scheduler wakes up nodes even when green pool nodes are available that could run that job. In other words, the nodes that are shutdown get a higher priority than the green pool nodes to run jobs.

To alleviate this problem, we have considered two variations of the algorithm mentioned above. The first one maintains a dynamic green pool, i.e. there are no fixed nodes in the green pool. The benefit of this approach is that, whenever possible, jobs do not have to wait until nodes get booted. While scheduling a job, the green pool nodes get a higher priority than the shutdown nodes in this algorithm but a lower priority than the non-green pool nodes similar to the Moab GC algorithm. The working of the algorithm is as follows. After having inspected the non-green pool nodes, if the job can be run on some green pool nodes, it is started on those nodes immediately. At the same time, the scheduler removes those nodes from the green pool and wakes up other nodes in the background which are added to the green pool. If there are no nodes that are in shutdown state, the scheduler adds some of the idle nodes (if possible) back to the green pool such that the size of the pool never changes.

The second variation is to simply give highest priority to the green pool nodes. Only if all the green pool nodes are busy, then the nodes that are shutdown and nodes from the non-green pool are considered for scheduling. As the green pool nodes are never shutdown,

they would be wasting energy if they are not doing any useful work for long periods. By running as many jobs as possible on those nodes, we are conserving energy as well as gaining more opportunities to shutdown the non-green pool nodes. Note that the underlying concept behind this algorithm is to try to maximize the number of shutdown nodes by running the available jobs on a minimum number of nodes (including the green pool nodes).

Algorithm:

Variant 1 - Moab-like DGP (Dynamic Green Pool):

For a job that requests N tasks,

1. Determine whether the job could be started on the active nodes in the cluster **not** belonging to the green pool. If yes, start the job.
2. If not, check whether the job could be started immediately on the nodes belonging to the green pool. If yes, start the job. Remove the nodes from the green pool and add an equivalent number of idle nodes back to the pool. If necessary, wake up some nodes.
3. If there are no nodes in the shutdown state, check if the job could be started using the green pool nodes.
4. If the job cannot be started after all these considerations, put the job in the waiting queue.

Algorithm:

Variant 2 - Moab-like Ideal GC (Ideal Green Computing):

For a job that requests N tasks,

1. Determine whether the job could be started on the nodes belonging to the green pool. If yes, start the job.
2. If not, check if the job could be started using the active nodes belonging to the non-green pool. If yes, start the job.
3. If not, check the nodes that are currently shutdown (if any) and wake them up if the job could be started using them. Start the job when the nodes have been booted up.

4. If the job cannot be started after all these considerations, put the job in the waiting queue.

In all of the algorithms discussed here, although not explicitly stated, they still use the energy aware scheduling approach used for the Packing Simulation to identify the least energy-consuming nodes whenever there are many nodes on which a job could be run. It is also assumed by the scheduling algorithm that CPU Packing is *not* being done on the individual nodes of the cluster.

4.6.2 Results

We performed simulations using the algorithms described above and using the log files. The results and analysis are given here:

4.6.2.1 Moab vs. Moab Green Computing

Table 4.3 compares Moab vs. Moab Green Computing with the green pool size equal to zero. It shows that Moab Green Computing provides an average of approximately 13% energy savings per week.

We can also see that the energy consumption of the cluster during a week is not proportional to the number of jobs submitted to the cluster. During the week of 1st April, a large number of small size jobs (small resource requirements and small walltimes) were submitted, but the effective load on the cluster was very low. Therefore, Moab Green Computing is able to shutdown a large number of nodes as needed and saves approximately 34% energy. For some weeks, although the number of jobs that were submitted was relatively small, the jobs had bigger resource requirements requesting a large number of nodes and cores, and/or bigger walltimes. Consequently, the effective load on the cluster was high providing fewer opportunites for shutdown.

During the Moab simulation, for most weeks, the maximum node usage in the cluster was between 90 and 115. This load indicates that about 10-40 nodes were not being used by Moab

Table 4.3: Energy savings: Moab vs. Moab Green Computing
(Number of nodes in green pool = 0)

Week start date	Number of nodes used	Jobs submitted during the week	Moab (kWh)	Moab Green Computing (kWh)	%Energy Savings	Cost Savings ($)
1st April	115	23,572	4,735.48	3,079.00	34.98	165.65
5th April	116	20,715	5,068.34	4,323.05	14.70	74.53
10th April	116	14,700	5,176.74	4,629.67	10.57	54.71
15th April	103	11,367	4,514.14	3,599.76	20.26	91.44
22nd April	105	11,820	4,616.89	3,739.83	19.00	87.71
29th April	106	12,264	4,728.41	4,036.90	14.62	69.15
5th May	92	11,911	3,980.82	2,882.93	27.58	109.79
12th May	106	12,526	4,795.84	4,396.05	8.34	39.98
15th May	106	3,722	4,811.33	4,464.28	7.21	34.71
22nd May	101	7,631	4,655.44	4,494.19	3.46	16.13
1st June	104	5,100	4,689.71	4,465.85	4.77	22.39
7th June	96	5,707	4,434.60	4,396.57	0.86	3.80
13th June	95	5,654	4,337.76	4,145.13	4.44	19.26
1st July	112	5,583	4,975.17	4,480.17	9.95	49.50
15th July	105	2,637	4,562.90	3,784.12	17.07	77.88
18th July	106	3,573	4,585.59	3,571.13	22.12	101.45
25th July	122	6,778	5,466.46	4,787.54	12.42	67.89
1st August	122	13,308	5,552.99	5,287.26	4.79	26.57

during the corresponding weeks for whatever reason and therefore, in all our simulations, we assume that those nodes are in the shutdown state. All the energy savings listed in Table 4.3 are relative to the maximum node usage during the corresponding week.

4.6.2.2 Energy comparison between the shutdown algorithms

Figures 4.7 and 4.8 show the energy consumption of the cluster as the number of nodes in the green pool is increased from 0 to 100 during a busy week and a less busy week. As we can see from the graph, during the busy week, Moab Green Computing provides approximately 60 kWh energy savings (equivalently $6 per week savings if the cost of energy is assumed to be 10 cents per kWh). On the other hand, during the less busy week, it could provide about

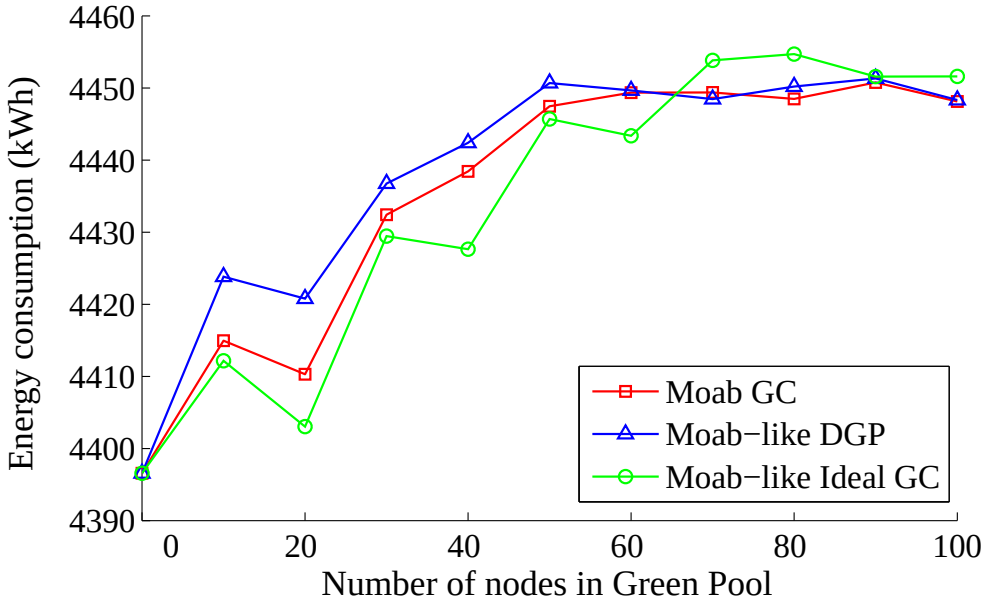

Figure 4.7: Simulation results for a busy week

650 kWh (sometimes more) energy savings (equivalently $65 per week savings).

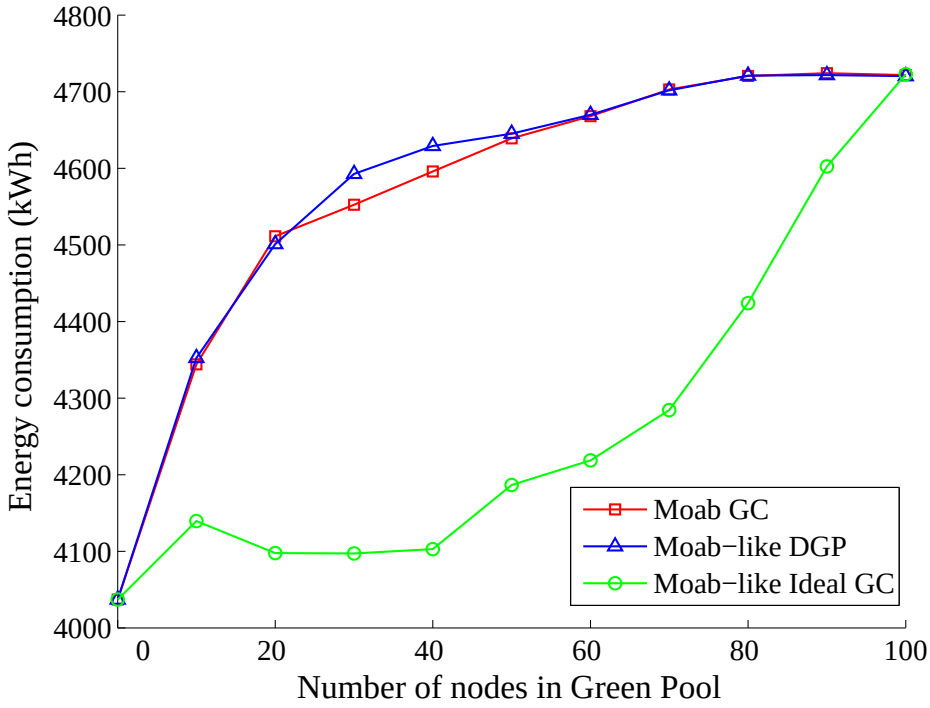

Figure 4.8: Simulation results for a less busy week

In both cases, the Moab-like Ideal GC algorithm consistently consumes less energy as compared to the other two algorithms. This savings is expected because in the Moab-like Ideal GC algorithm, the nodes in the green pool are favored before those that are shutdown or that belong to the non-green pool. However, we believe that the Moab scheduler with Green Computing enabled on the real cluster would behave like the red plot (with square markers) in the graphs, since it represents the Moab GC algorithm that we have deduced based on our experiments with the Green Computing feature on our test cluster.

In all cases, the cluster consumes the least energy when the green pool is empty. This is expected due to the fact that as the number of nodes in the green pool tends to zero, the scheduler has more opportunities for shutdown.

We also did a back-of-the-envelope calculation to find out the energy consumption of the cluster using Moab simulation under the assumption that a node is shutdown immediately whenever it becomes idle and it is started instantly the moment it starts running some job. We compared it with the energy consumption for Moab Green Computing with green pool size equal to zero. For most weeks, the energy consumption of the Moab simulation was within 1-2% (sometimes less, sometimes more) of that of Moab with Green Computing, while for some weeks it was within 2-7% (again, sometimes Moab has better energy savings and sometimes our simulation). However, since the calculation was a first order approximation in the first place and also that the energy consumption of the cluster using Moab would only increase if we consider the power consumption of nodes during the time they are idle, we think that Moab would provide almost same energy savings as the Moab Green Computing simulation.

4.6.2.3 User pain involved in shutting down nodes

Although shutting down nodes provides substantial savings in energy, it also has some side effects. Some jobs might have to wait longer in the queue before they get serviced while the shutdown nodes get booted. For jobs that run for many hours or days, this is not a problem

as an additional waiting time involved in waking up nodes is insignificant. However, for short duration jobs, larger wait times might cause inconvenience to the end users. Therefore, it is important to separate the jobs into different classes and measure the user pain for each class. For this purpose, we divide the jobs in the cluster into three classes:

a) small size jobs having a run time of less than one hour.

b) medium size jobs having a run time of more than one hour but not more than a day.

c) large size jobs having a run time of more than one day but not more than one week.

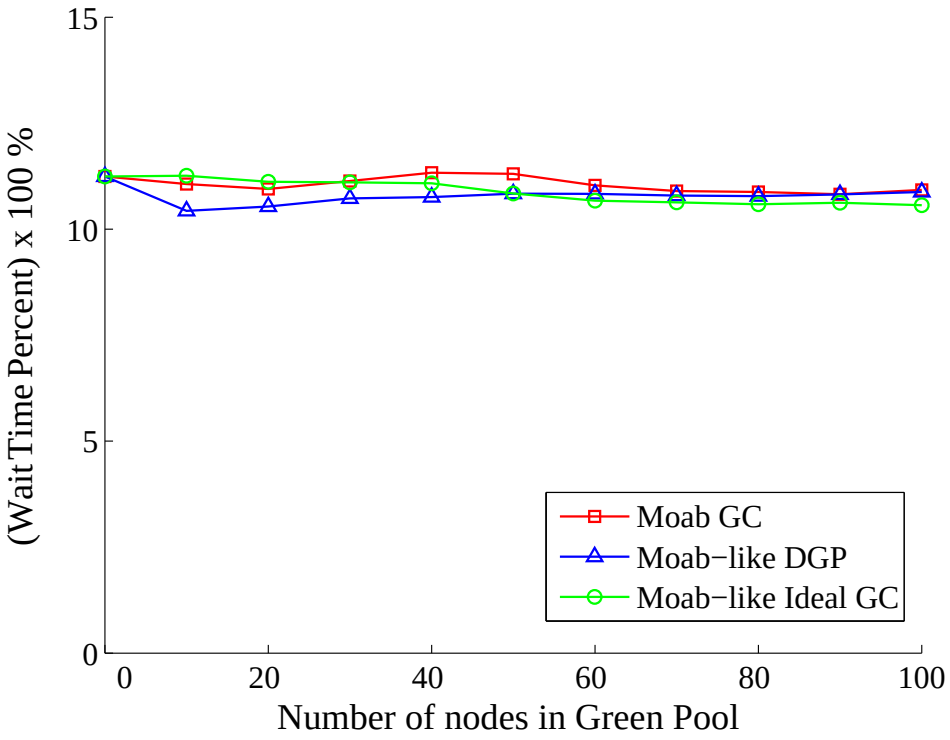

Figure 4.9: Pain corresponding to number of nodes in green pool

For measuring the pain, the wait time of a job alone is not a good metric. Consider two jobs that have the same wait time of 30 minutes, but one runs for an hour whereas the other runs for a day. The user of the shorter job experiences much more pain as compared to the user of the longer job. This difference shows that while characterizing pain, it is important to consider the run time of a job along with the wait time. Therefore, we use the ratio,

$WaitTimePercent = (Wait\ time\ of\ job)/(Turnaround\ time\ of\ job)$,

as our pain metric, where the turnaround time of a job is just the sum of its waiting time and actual run time.

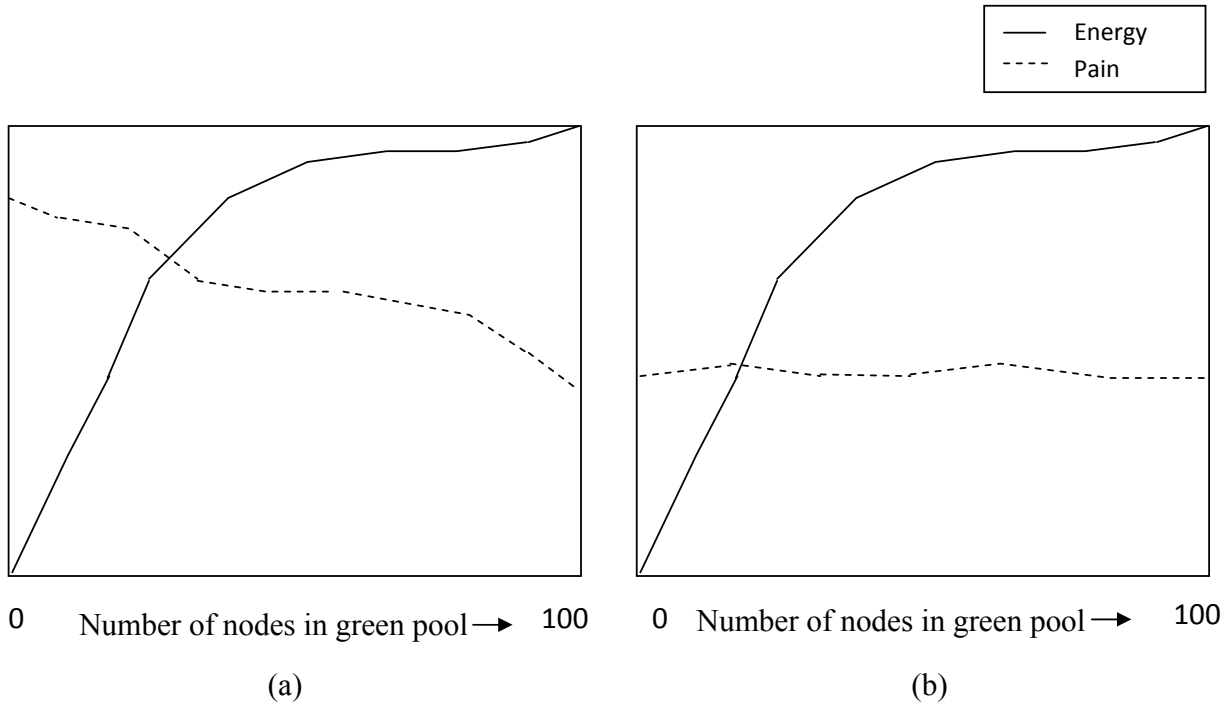

Figure 4.10: Decreasing pain vs. Constant pain

The pain plot for small size jobs using our metric is shown in Figure 4.9. The plots for the other classes of jobs are similar. As can be seen in the figure, the pain (given by $WaitTimePercent * 100$) remains constant as the number of nodes in the green pool is varied from 0 to 100. Before the simulations, we were expecting that as the number of nodes in the green pool is increased, the pain for the users would decrease as shown in Figure 4.10(a). If that were to be true, we would have been able to approximate the optimal number of nodes to be maintained in the green pool on average that would minimize energy consumption of the cluster as well as the pain to the users. However, since the simulations show that pain is almost constant (as in Figure 4.10(b)) regardless of the number of nodes in the green pool, the best choice seems to be a green pool of size zero as it minimizes the energy consumption of the cluster.

Theoretically, it might be a good idea to keep the green pool empty such that all the

nodes in the cluster could be shutdown. However, we could think of a scenario where all the nodes in the cluster are shutdown and many jobs are submitted to the cluster at once. Since there are no active nodes available to service the jobs, the jobs would have to wait in the queue until the scheduler wakes up nodes. To avoid this and to give better service to the users, a small number of nodes (approximately 10) should be maintained in the green pool.

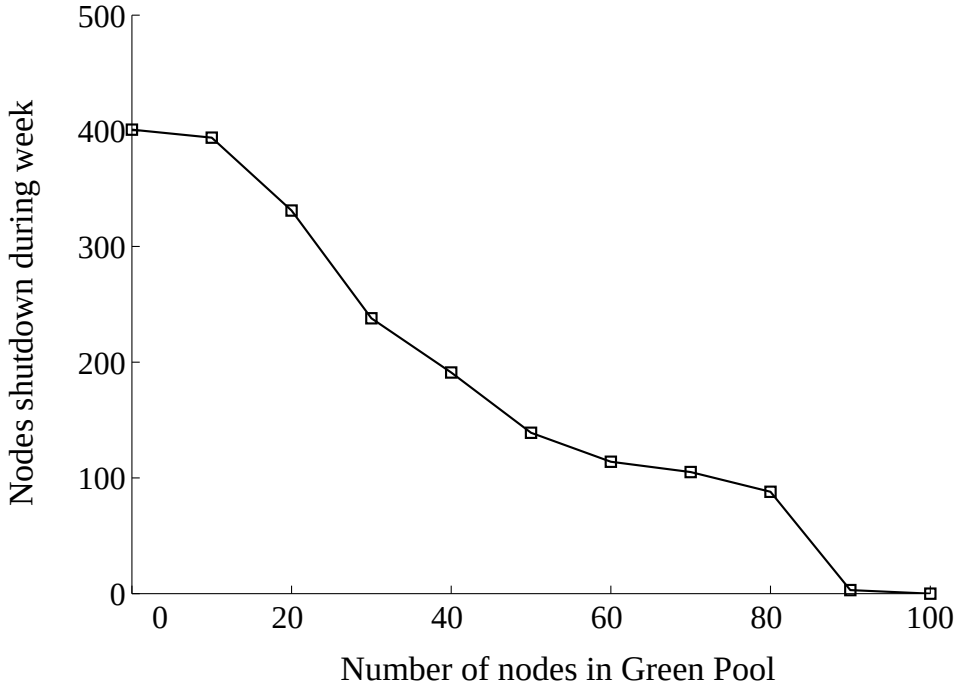

Figure 4.11: Number of shutdowns corresponding to green pool size

From the point of view of the cluster administrators, excessive cycling of nodes is not desired. The simulations indicate that as the number of nodes maintained in the green pool is increased, the number of node startups/shutdowns in the cluster decreases. Figure 4.11 is a typical graph for most weeks showing the number of shutdowns during the week for different number of nodes in the green pool. On an average, when the green pool size is zero, the number of startups/shutdowns is approximately 400, i.e. if the nodes are shutdown uniformly during the week, each node would be shutdown and/or restarted four times. As a policy, the administrator may choose the number of nodes to be maintained in the green pool based on the maximum number of shutdowns that are allowable per week.

4.7 Recommendations to the HPC Administrators

1. Turn on DVFS : Install *"cpufreq"* module, if not already installed. In the BIOS, enable the Enhanced Intel Speedstep Technology (EIST) option. This will cause the CPU to run at the lowest frequency when it is not doing any work and save energy.
2. Turn on CPU Packing : Enable the feature by writing 1 to, *"/sys/devices/system/cpu/scheduler_mc_power_savings"*.

   If Linux considers a node to be busy, it might not do packing even if enabled. However, if the load is less, Linux performs packing on the node resulting in lesser power consumption by the node.
3. Enable Green Computing: Upgrade Moab to v5.3.5+ (the current version of Moab on the cluster is v5.3.4). Configure Green Computing parameters in the "moab.cfg" file. A node startup/shutdown script has to be provided to Moab to make Green Computing work. The script could use IPMI commands although Wake-On-Lan seems to be sufficient.

4.8 Comparison with the current cluster

The logs used for the simulation represent the cluster that existed two years ago (in 2008). If the effect of Green Computing on the cluster as it exists today has to be measured, new logs have to be collected and the simulations have to be run using those logs. However, we believe that the results of this thesis, even with the older logs, would still hold for the current cluster.

We know that, in 2008, the Intel and AMD clusters had separate queues; whereas now, there is a common queue for both the clusters. Basic investigations over a period of two weeks show that the number of jobs being submitted to the common queue currently per week is approximately 20,000. So, we could assume that out of the 20,000 jobs, the Intel cluster runs about 14,000 jobs (since the number of cores in the Intel cluster is twice that of the AMD cluster, we assume that about two-third of the total load would be handled by

the Intel cluster). A large number of those jobs had used just one or two nodes and one to four cores per node. The logs used for the simulations show that many weeks during April, May and August 2008 had a similar load, viz. approximately 12,000-20,000 jobs were submitted during those weeks and the characteristics of those jobs are similar to that of the jobs examined in the current cluster. The Green Computing Simulation shows energy savings in the range 5-20% for those weeks (refer Table 4.3). Therefore, we expect that Moab Green Computing would provide energy savings even in the current cluster.

Chapter 5

Conclusions and Future work

5.1 Conclusions

1. We investigated the various power saving techniques available on servers. CPU Packing could reduce the power and energy consumption of a node if the jobs to be run are mostly CPU-bound (we consider CPU-bound jobs to be those that have small memory usage or even if they have a higher memory usage, they additionally have a good cache locality); whereas for jobs with a higher memory usage but poor cache locality, it could decrease the performance by a lot. The intra-node power saving techniques like, CPU Packing and DVFS, provide only nominal energy savings whenever possible. Shutting down a node provides maximum power savings as expected.

2. We implemented an energy-aware cluster scheduling algorithm that assumes that CPU Packing is enabled on the cluster nodes. The results of the simulation suggest that the behavior of the proposed algorithm is very similar to the one used by Moab (both of them perform node level packing) and doesn't provide any substantial energy savings on its own.

3. We experimented with Moab Green Computing on a test cluster of four nodes. Based on our observations, we performed a simulation to determine its effect on the actual HPC cluster. The results show that Moab Green Computing could provide on an average 13% energy savings for the HPC cluster during a week when the green pool size is zero, without any noticeable decrease in the performance of jobs. We expect the Moab scheduler (with Green Computing enabled) in the HPC cluster to behave like our simulation; but due to the limitations of the logs used for the simulation as well as the various other

policies used by Moab but not considered for the simulations, its exact behavior on the HPC cluster cannot be entirely predicted. A variant of the Moab Green Computing simulation was shown to perform slightly better in terms of energy savings (with the same number of nodes in the green pool in both cases) than the Moab Green Computing simulation.

5.2 Future work

1. In our simulations, we have assumed that CPU Packing does not affect the performance of the jobs. However, in practice, it does decrease the performance (for some jobs, the decrease is very small; whereas for others, it can be large). To the best of our knowledge, the theory related to CPU Packing and its effect on the performance and energy of jobs has not been investigated before. We think that there is scope for some theoretical research here.

2. In our simulations, we have noticed that the time duration between a node shutdown and the next node startup, or vice versa, is small (less than 10 minutes) on many occasions. By doing some intelligent scheduling and not using a static policy for the node idle duration (after which a node is shutdown), we think that the number of node shutdowns and startups in the cluster could be reduced while achieving similar energy savings. This needs to be further investigated.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Advanced Configuration and Power Interface. *http://www.acpi.info/*.

[2] GANGLIA Monitoring System. *http://ganglia.sourceforge.net/*.

[3] IBM Tivoli Loadleveler Scheduler. *http://www-03.ibm.com/systems/software/load leveler/*.

[4] Moab Cluster Scheduler. *http://www.clusterresources.com/products.php*.

[5] NAS Benchmark Suite. *http://www.nas.nasa.gov/Resources/Software/npb.html*.

[6] SPEC CPU2000 Benchmark Suite. *http://www.spec.org/cpu2000/*.

[7] Watts Up Pro! Power Meter. *https://www.wattsupmeters.com*.

[8] Datacenter energy costs outpacing hardware prices. *http://arstechnica.com/business/news/2009/10/datacenter-energy-costs-outpacing-hardware-prices.ars*, 2009.

[9] Moab Workload Manager Administrator's Guide v5.4. 2010.

[10] Powersave Algorithm: Rocks Solid Package. 2010.

[11] US Energy Information Administration. Count of Electric Power Industry Power Plants, by Sector, by Predominant Energy Sources within Plant. *http://www.eia.doe.gov/cneaf/electricity/epa/epat5p1.html*, 2010.

[12] Environmental Protection Agency. Report to Congress on server and Data Center Energy Efficiency Public Law 109-431. *http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf*, 2007.

[13] Susanne Albers, Fabian Muller, and Swen Schmelzer. Speed scaling on parallel processors. In *Proc. SPAA (2007)*.

[14] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *HANDBOOK ON SCHEDULING ALGORITHMS, METHODS, AND MODELS*. Chapman Hall/CRC, Boca, 2004.

[15] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[16] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient Server Clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, pages 179–196, 2002.

[17] Yoav Etsion and Dan Tsafrir. A Short Survey of Commercial Cluster Batch Schedulers.

[18] Dror G. Feitelson and Ahuva Mu'alem Weil. Utilization and predictability in Scheduling the IBM SP2 with Backfilling.

[19] Vincent W. Freeh, Tyler K. Bletsch, and Freeman L. Rawson III. Scaling and Packing on a Chip Multiprocessor. *ipdps, pp.349, 2007 IEEE International Parallel and Distributed Processing Symposium*, 2007.

[20] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L. Rountree, and Mark E. Femal. Analyzing the Energy-Time Trade-off in High-Performance Computing Applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):835–848, 2007.

[21] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Energy-efficient scheduling of HPC applications in Cloud Computing Environments. *Preprint submitted to Elsevier*, 2009.

[22] Soraya Ghiasi and Wes Felter. Cpu Packing for Multiprocessor Power Reduction. *Springer-Verlag Berlin Heidelberg: Lecture Notes in Computer Science*, 2004.

[23] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal 45: 15631581*, 1966.

[24] Saeed Iqbal, Rinku Gupta, and Yung-Chin Fang. Planning Considerations for Job Scheduling in HPC Clusters. *http://www.dell.com /downloads/global/power/ps1q05-20040135fang.pdf*, 2005.

[25] Sandi Irani and Kirk R. Pruhs. Algorithmic Problems in Power Management.

[26] David Jackson, Quinn Snell, and Mark Clement. Core Algorithms of Maui Scheduler.

[27] J.K.Lenstra and A.H.G.Rinooy Kan. An Introduction to Multiprocessor Scheduling. *Technical Report, CWI, Amsterdam*, 1988.

[28] Andre Kerstens and Steven A. DuChene. Applying Green Computing to clusters and the data center. *Proceedings of Linux Symposium*, 2008.

[29] Trevor Mudge. Power: A First-Class Architectural Design Constraint. *Computer*, 2000.

[30] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems, 2001.

[31] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Dynamic Cluster Reconfiguration For Power and Performance, 2002.

[32] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines online. *IEEE*, 1991.

[33] Suresh Siddha, Venkatesh Pallipadi, and Asit Mallick. Chip Multiprocessing aware Linux Kernel Scheduler. *Intel*, 2005.

[34] Suresh Siddha, Venkatesh Pallipadi, and Asit Mallick. Process Scheduling Challenges in the Era of Multi-core Processors. *Intel Technology Journal*, 11(4), November 2007.

[35] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and P. Sadayappan. Characterization of Backfilling Strategies for Parallel Job Scheduling.

[36] US Energy Information Administration Independent Statistics and Analysis. Net generation by energy source. *http://www.eia.doe.gov/cneaf/ electricity/epm/table1_1.html*, 2010.

[37] Frances Yao, Alan Demers, and Scott Shenker. A Scheduling Model for Reduced CPU Energy. *IEEE*, 1995.