


2015

Energy Agile Cluster Communication

Muhammad Zain Mustafa
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/masters_theses_2

 Part of the [Data Storage Systems Commons](#), [Hardware Systems Commons](#), [Power and Energy Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

Mustafa, Muhammad Zain, "Energy Agile Cluster Communication" (2015). *Masters Theses*. 164.
https://scholarworks.umass.edu/masters_theses_2/164

This Open Access Thesis is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Masters Theses by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

ENERGY AGILE CLUSTER COMMUNICATION

A Thesis Presented

by

M. Z. MUSTAFA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2015

Electrical and Computer Engineering

ENERGY AGILE CLUSTER COMMUNICATION

A Thesis Presented

by

M. Z. MUSTAFA

Approved as to style and content by:

David E. Irwin, Chair

Tilman Wolf, Member

Michael Zink, Member

C.V. Hollot, Department Chair
Electrical and Computer Engineering

ABSTRACT

ENERGY AGILE CLUSTER COMMUNICATION

FEBRUARY 2015

M. Z. MUSTAFA

B.Sc., NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
(PAKISTAN)

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor David E. Irwin

Computing researchers have long focused on improving energy-efficiency—the amount of computation per joule—under the implicit assumption that all energy is created equal. Energy however is not created equal: its cost and carbon footprint fluctuates over time due to a variety of factors. These fluctuations are expected to intensify as renewable penetration increases. Thus in my work I introduce energy-agility, a design concept for a platform’s ability to rapidly and efficiently adapt to such power fluctuations. I then introduce a representative application to assess energy-agility for the type of long-running, parallel, data-intensive tasks that are both common in data centers and most amenable to delays from variations in available power. Multiple variants of the application are implemented to illustrate the fundamental tradeoffs in designing energy-agile parallel applications. I find that with inactive power state transition latencies of up to 15 seconds, a design that regularly “blinks” servers outperforms one that minimizes transitions by only changing power states when power

varies. While the latter approach has much lower transition overhead, it requires additional I/O, since servers are not always concurrently active. Unfortunately, I find that most server-class platforms today are not energy-agile: they have transition latencies beyond one minute, forcing them to minimize transition and incur additional I/O.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER	
1. INTRODUCTION	1
1.1 Energy-Agility	3
1.2 Background	5
1.3 Problem	7
1.4 Solution	9
1.5 Hypothesis	10
1.6 Contributions	11
2. RELATED WORK	14
2.1 Scheduling Policy Design	15
2.2 Energy Proportional Design	19
2.3 Higher Renewable Energy Penetration	21
3. ENERGY AGILE COMMUNICATION	29
3.1 Emerging Scenarios	31
3.2 All to All Communication	32
3.3 Application Design	34
3.3.1 Distributed Environment	36
3.3.2 Policies	39
3.3.3 Design Concepts	40
3.4 Energy Agile Policies	42

3.4.1	Active Power Cap	42
3.4.2	Burst	43
3.4.3	Blink	45
3.4.4	Battery Burst	47
3.5	Power Input	48
4.	IMPLEMENTATION	50
4.1	Theoretical Verification Through Controlled Simulation	50
4.1.1	Policy Development Testbed	51
4.1.2	Policy Evaluation Tool	52
4.1.3	Simulator Design	52
4.1.4	Simulator Output	53
4.2	Application Development And Deployment	58
4.3	Hardware Power States Profiling	60
4.3.1	C States	60
4.3.2	P States	62
4.3.3	Controlling CPU Cores	63
4.3.4	Throttling Disk IO and Network Bandwidth	65
5.	EVALUATION	67
5.1	Sequential Write With Rapid S3 Transitions	67
5.2	Energy Agile Performance	68
5.3	Effect Of Transition Time	72
5.4	Effect Of Energy Buffer Size	75
5.5	Impact of power variation	77
5.6	Effect of Job Size	78
5.7	Experiment Results	80
6.	CONCLUSION	82
	BIBLIOGRAPHY	84

LIST OF TABLES

Table	Page
3.1 C State Details	38

LIST OF FIGURES

Figure	Page
1.1 Computational Ability Growth	2
1.2 Operational Costs for Generators	4
1.3 Renewable Energy Variation	5
3.1 All to All Internode Communication	34
3.2 Simplified Cluster Diagram	36
3.3 Balanced vs Unbalanced Links	41
3.4 Sudden full throttle power	42
3.5 Burst Transitions	44
3.6 Blink Interval	46
3.7 Battery Burst	47
3.8 Wind Power Signal	48
3.9 Solar Power Signal	49
4.1 Simulations with different power budgets	55
4.2 Simulations with different transition times	56
4.3 Simulations with different buffer size	57
4.4 Massachusetts Green High Performance Computing Center	59
4.5 C States Energy Consumption	61
4.6 C States Latency	62

4.7	P States Energy Consumption	63
4.8	Effect of CPU Cores on Power	65
4.9	Effect of Power on Network and Disk IO throttle	66
5.1	Effect of S3 transitions on Write a Sequential File	68
5.2	Power Budget Performance	70
5.3	Power Budget Performance	72
5.4	Transition Time Analysis	73
5.5	Transition Time Wind Analysis	73
5.6	Transition Energy Analysis	74
5.7	Transition Energy Wind Analysis	75
5.8	Energy Buffer Size Variation	76
5.9	Impact of varying power on policy performance	77
5.10	Impact of varying power on policy energy	78
5.11	Impact of job size on policy performance	79
5.12	Impact of job size on policy energy	80

CHAPTER 1

INTRODUCTION

Data centers today represent a significant chunk of computing power. They are responsible for entire online infrastructures of companies, for carrying out experiments and scientific research among other things. They also form the foundation of the internet by providing services such as cloud computing, hosting websites etc. They are typically large facilities used to house computing systems and supporting devices. Depending on their size, they can house anywhere between hundreds to hundreds of thousands of networked computer servers along with massive data storage. As each server draws a significant amount of power, the total energy demands at data centers can be massive. Additional power must also be used to power the cooling systems and other devices housed in the data center. Depending on the size of the facility, a data center can have energy demands similar to that of a small town. Hence many data centers are equipped with their own back up power supplies and may also house flywheels or other form of energy storage. It is estimated that for high density data centers, the Total Cost of Ownership (TCO) for energy consumption can be as high as 30% [18]. Such high levels of energy consumption can also have significant impacts on environmental emissions. Data centers therefore aim to reduce their energy consumption and it's consequent environmental impact.

However data centers deal with a highly competitive industry and must be able to deliver performance of the order of state of the art technology. While reducing environmental emissions and dealing with high level of energy consumption, it is important for data centers to also aim at performance growth. This increasing per-

formance growth (figure 1.1) implies a similar increase in energy consumption used by the servers. Energy consumption for High Performance Computing (HPC) servers has increased to the extent that power considerations are now amongst the major design constraints for the next class of HPC platform exa-scale supercomputers. Hence data centers look towards more cost effective energy sources.

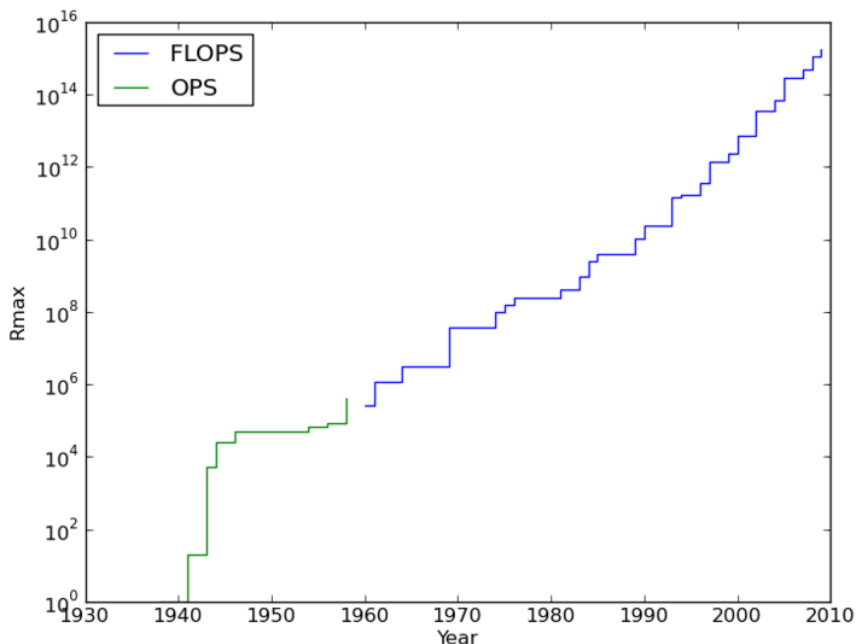


Figure 1.1. Computational Ability Growth

For these reasons and more explained below, data centers have begun to shift towards using renewable power sources and participate in demand response programs initiated by the power utilities. Both these factors encourage the data centers to adapt to highly fluctuating input power. Renewable power has an inherent unpredictability in almost every renewable power source. Although solar power can be predicted to some level of accuracy, it is better for energy agile designs to assume all sources are unpredictable so that system can respond to any kind of stochastic power signal, being "power signal agnostic". On the other hand, data centers participating in utility

initiated Demand Response programs have to respond to signals from the utilities to increase or decrease their power demands as the power supply varies. The participants are typically compensated based on their response time and the capacity by which they can ramp up their power demands.

In this work, we term this ability to carry out typical data center tasks under fluctuating input power as Energy Agility. Interestingly, although data centers typically possess the capabilities to respond to fluctuating power in their individual servers such as power capping, using Dynamic Voltage Frequency Scaling (DVFS) states (P-States), inactive cpu power states (C-States), inactive ACPI power states (S3,S4), etc; they lack the ability at the cluster level to leverage these individual server capabilities to control the overall cluster power demand. In this work, we introduce the concept of energy agility for clusters and show how cluster wide policies for server operations can be used to enable data centers to carry on their work even when input power is not stable. Energy agility is defined as follows.

1.1 Energy-Agility

Energy Agility is a similar concept to energy efficiency. With respect to computing, energy-efficiency is defined as the amount of work, e.g. computation, I/O, etc., done per unit of energy. For sorting, energy-efficiency translates to the number of records sorted per joule (or watt-second) *of energy consumed by a platform*. Energy agility however can be more easily understood in the same context by the amount of work done given a power budget which may vary over time. Thus similarly for sorting, energy-agility translates to the number of records sorted per joule *of energy given to a platform* independent of how much energy the platform consumes. This definition can be extended to data shuffling as well where energy-agility would become the number of records shuffled per joule of energy given to a platform.

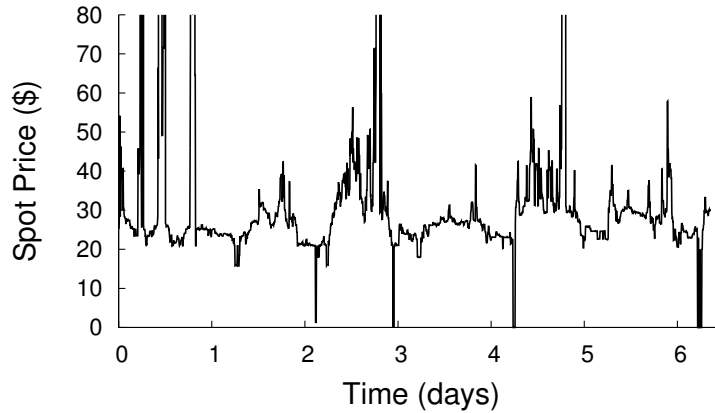


Figure 1.2. Operational Costs for Generators

Energy agility although similar in concept to energy efficiency, is agnostic to how efficient a platform is. It encourages applications to use all available energy whenever possible, applying a "use it or lose it" property. Of course the applications are also independently motivated to use the available energy as efficiently as possible. Unused energy can be stored assuming there is some form of energy storage mechanism available and later used for work. I consider the impact of energy storage capacity on energy agility in my work in later sections. Power budgets however must be scaled to better suit the working platform. For example, a power budget that periodically provides more energy than the platform's maximum consumption forces it to waste the excess energy if there is no energy storage available for that time period.

Although the cost of producing energy and its carbon footprint is highly dependent on the power source, energy consumption is ignorant of these costs. Energy efficiency optimizations aim for doing more work using less energy regardless of the power source. In contrast however, energy agility reflects the fact that all energy is not created equal [30]. Its cost can instead be driven from a heterogeneous mix of generators with different fuel costs, carbon emissions and operational characteristics. As an example, solar and wind energy are created without any fuel costs or carbon emission. However the generation in this case is dependent not the environment con-

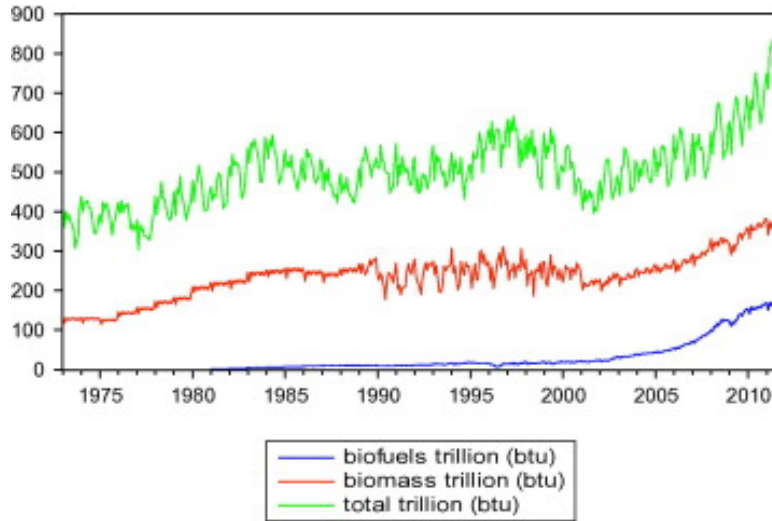


Figure 1.3. Renewable Energy Variation

ditions. On the flip side however, "peaking" generators are utilized by utilities to satisfy excess power demand on the grid. These generators are capable of producing energy at demand but with a high fuel cost ($\approx 10x$) and environmental emissions [9, 33]. Figure 1.2 shows the different marginal cost of energy produced from various sources. The figure illustrates the disparity in the high cost of producing energy using any form of generator and the unstable fluctuations possible in renewable energy which together contribute to energy price fluctuations over time. As we look towards more renewable penetration, power fluctuations may become a permanent feature of power supply [20] causing more intense energy price fluctuations. Figure 1.3 on the other hand shows the variations in renewable energy produced every month over the years since 1973. The figure shows a seasonal trend in the amount of total renewable energy produced over the years.

1.2 Background

Data centers have experienced improvements in efficiency in the operational use of energy over the recent years. As of 2010, the average Power Usage Effectiveness(PUE)

index in the U.S data centers was 2.0 with the best PUE around the scale of 1.2 by Google data centers [2]. However despite these improvements, the cumulative energy consumption of data centers around the world have increased over the same time period. Worldwide data center energy demands have been estimated to have increased by 56% between 2005 and 2010. In the U.S alone data centers account for 1.7-2.2% of the total electricity usage nationwide [17]. These increasing energy demands have prompted leading data centers to cut down the energy carbon footprint and cost. Current state of the art data centers such as the Massachusetts Green High Performance Computing Center (MGHPCC) are now experimenting with local renewable power sources as an alternate source of energy [1, 10, 24]. Commercial companies such as Google and Apple have also joined the charge with Apple now claiming that it's data centers are run on 100% renewable energy. Besides using renewable energy, data centers have also started participating in utility demand response (DR) programs. As mentioned, both renewables and DR introduce the potential for variations in a data center's available power.

Data centers are well equipped to respond to such power variations as has been noted before [16, 34]. High end servers used by data centers already include sophisticated power management functions, which are capable of varying their power usage over a wide dynamic power range. They also often execute non-interactive batch applications that are tolerant to some delays due to power shortages.

However, although there has been research focused on improving the energy-efficiency, i.e. the amount of computation per joule of hardware and software, there has been much less research on *energy-agility*: the ability to adapt applications to dynamic variations in available power. Where energy efficiency seeks to minimize the amount of energy used to do some work, energy-agile systems focus on maximizing the amount of work done subject to available power. More specifically, energy agility in a cluster would mean that the performance of an application running on the cluster

can change with and respond to changes in input power to the cluster without actually disrupting the work being done by the application. The application may slow down or speed up with power changes, but it would continue operation.

Data centers commonly deal with long-running, massively parallel and data intensive batch applications that are amenable to delays from variations in input power. Unlike shorter batch job, such "big data" applications are not suitable for simply deferring their execution until enough power is available [13, 14]. Since they take a long time to run and complete, they must adapt to power fluctuations during execution rather than wait for an ideal uninterrupted time period with sufficient power which may never arrive. As with energy efficiency, an application's software design and its hardware capabilities contribute to its energy agility. In my thesis, I analyze the effect of both by introducing an energy agile internode data shuffling application.

1.3 Problem

To reduce costs and environmental emissions, data centers are now finding themselves more deeply integrated with renewable energy sources and involved in greater participation in demand response programs initiated by power utilities. Data center applications therefore are faced with the challenge of running on volatile power sources which can vary with environmental conditions (Renewable Energy) or power demand on the grid (Demand Response). Data center applications are typically long running massively parallel "big data" applications which can not be simply deferred until stable sufficient power is available to run them. They therefore need mechanisms embedded in their software design which leverage hardware capabilities of the servers in data centers to adapt to changing input power. Without the assumption of an extreme large energy storage mechanism which can support the servers running the application uninterrupted until it completes, running such applications is a challenge with the typical setup and design of current data center applications. However such

large energy buffers are not very cost effective and can be unpractical if an application runs for several days before completion.

As mentioned, data centers typically deal with massively parallel distributed applications such as Dryad, PIG, Map Reduce, Distributed Sort, etc. All these applications generally involve large amounts of data movements across the nodes in the clusters and data centers. Interestingly such cluster based applications are not usually bottlenecked by the CPU, but by disk speeds since they involve a large amount of data transfers. A large amount of nodes are typically involved in such large scale parallel applications with each node having to communicate with other nodes to send or receive data resulting in a data shuffle across all the nodes. For most distributed parallel applications, some form of data shuffle is an integral part of the application itself. Data shuffling or other forms of inter node communication in the cluster make a distributed parallel application different from other sequential jobs. It is interesting to note however, that the individual workloads of the nodes which are running a distributed parallel application can be independent relative to other nodes except for inter node data transfers and communications. The problem for data shuffling however arises when there is insufficient power to support all node which have to shuffle data amongst themselves. If a subset of nodes is powered with other nodes in some form of inactive state consuming negligible power, the powered nodes might transfer data amongst themselves but will not be able to share their data with the nodes which are inactive.

While data shuffle is admittedly a narrow problem just as sorting is, previous benchmarks based on similar problems have influenced a broad range of systems. For instance, the notion of balance in JouleSort [28] influenced the design of energy-efficient key-value stores [4], MapReduce platforms [27] and databases [32]. Data shuffling is also a common subtask for parallel applications often; often forming the

bottleneck for such application. Hence it captures the bottleneck for most data centric applications which are data intensive and parallel in nature.

Data centers therefore are faced with the challenge of running distributed applications involving data shuffling amongst nodes with varying input power. Such applications must have energy agile software designs to enable the data center to productively use varying energy. Without a large enough energy buffer to support all nodes until the application completes execution, this energy would otherwise be wasted.

1.4 Solution

In order to deal with variable power input, data center software must be designed to be energy agile. There has been previous work which uses sorting as a basis for benchmarks to highlight various aspects of system design such as I/O performance, energy efficiency and cost to motivate researchers to improve upon them. My work however adapts data shuffling as the core application as it is a subtask of almost every distributed application including sorting. Data shuffling also captures the basic principle of distributed applications i.e communication between nodes. Since it requires all the nodes involved in the shuffle to be up and running in order to complete work, it makes for a good problem to solve under variable power where all nodes may or may not be able to run together. I use data shuffling to highlight important software designs which can prove useful in designing energy agile applications for researchers in the future.

I design my application with energy agility requirements such that it is able to run and complete on a cluster with variable power by utilizing the hardware capabilities of nodes such as inactive power states (ACPI S3). As power fluctuates, some nodes in the cluster can be shifted to inactive hardware states where they consume negligible power to satisfy the current power budget. These active and inactive hardware power states

can be exploited through hardware management policy designs to allow energy agile operation in the data shuffling application. However such policies add a new dynamic to the data shuffling problem. In addition to minimizing the completion time of data shuffling, these designs must consider the effect of inactive power state transitions, which are time consuming and may periodically render some data unavailable. As with prior work involving sorting benchmarks, part of my work is aimed to distill a set of design principles capable of influencing a broader range of energy-agile applications. For my work, these design principles are a byproduct of the inactive power state management policies that I have designed.

To evaluate the software designs for my distributed shuffle application before implementation, I have developed a simulator where I can test my designs. The simulator works as a testbed for evaluating algorithms under different conditions. Since the simulated environment is completely controllable, it can also be a valuable tool to determine which of the policies would perform best under a certain set of conditions. Designs and simulations however can only be verified through actual implementations. Hence each hardware state management policy is implemented within the framework of my data shuffle application and has been deployed and tested on the MGHPCC cluster.

1.5 Hypothesis

I have designed four different hardware state management policies which allow my data shuffling application to operate with energy agility. Each of these policies makes it possible for my application to continue running even with fluctuating input power. However each policy is designed to work well for a mix of conditions and assumptions such as inactive hardware state transition time, how much power is available on average, the extent of input power fluctuation and the amount of energy storage available in the cluster.

Performance of the application with different hardware management policies can be measured by the completion time of the data shuffling application for a given set of conditions. Since we want the applications to complete as fast as possible, a smaller execution time indicates better performance. Policy performance under certain conditions can therefore be measured by running the data shuffling application with all the policies. The least completion time in this case would indicate the best performance for those conditions.

Since each policy is designed to work well for different conditions such as the amount of energy storage available and the transition time for going in and out of the inactive hardware states for nodes, I expect each policy to perform better under their ideal conditions. The four policies I have designed are : Active power cap, Burst, Burst with battery and Blink. The latter two need energy storage mechanisms in order to work. These policies can therefore be categorized into storage aware and non storage aware policies. I expect that for practical conditions of large transition times between hardware inactive states, the Burst with battery policy which combines a non storage aware policy with an energy storage mechanism should perform the best. For other conditions where we assume we can invest in a large enough energy storage buffer and the transition times are not very large, Blink which is a storage aware policy should perform better.

1.6 Contributions

An important goal in developing an energy-agile data shuffling application is to identify design principles as important lessons for a larger host of distributed applications which can benefit from energy-agility. In addition to software developers considering energy agility as a design criteria for applications, another goal to motivate hardware vendors to incorporate functionality which facilitates energy agility such as faster transitions into ACPI S3 states. As scenarios which value energy

agility are only now emerging, largely due to both the decreasing cost of enable energy sources advances in smart grids, energy-agility has not been previously thought of as an important design metric. As a result, current hardware/software platforms are not particularly energy-agile, especially for I/O bound workloads that cannot fully utilize the active power states available on CPUs. Contributions of my work, through my thesis are:

Energy Agility Policies. Having established energy agility as a performance metric for distributed applications distinct from energy efficiency, I go on to define energy-agile software designs in the form of hardware state management policies. Although these policies are designed within the framework of my data shuffling application, they can be implemented in other distributed systems such as sort, map reduce, dryad, etc with ease. Since the policies have more to do with hardware management than with the software itself, they are easy to incorporate for any data center application.

Simulator. Another contribution that I make in my thesis is the development a simulator capable of simulating all my hardware management policies in a controlled virtual environment. This can serve as a test bed for developing more policies for energy agility and for identifying which policy would work better under a certain set of conditions.

Implementation and Evaluation. Finally to verify my simulator outputs and evaluate the actual performance of my policies, I have developed a framework data shuffling application which can run on an actual cluster with fluctuating input power. This framework can work with my hardware state management policies for an energy agile software design. All of my policies have been implemented and evaluated on the MGHPCC server under various conditions of hardware state transition time, energy storage availability and power fluctuations. This implementation has also

been evaluated against realistic power fluctuations adapted from real wind speed and solar emission traces.

CHAPTER 2

RELATED WORK

Energy Agility in the context of the amount of work done, given a power budget which may vary over time is not a very deeply explored research area. As mentioned in the introduction, we take on energy agility as a distinct but similar concept to energy efficiency where we try to use as much energy as possible whenever possible, leaving the energy efficiency details to be platform dependent. There has been a lot of previous and ongoing research work dealing with energy efficiency as researchers deal with the perils of high energy costs and rising environmental emissions from data centers. Fluctuating input power to data centers with higher renewable penetration and participating in demand response programs are recent developments for data centers and possible scenarios where energy agility would be useful however are only now emerging. Hence energy agility has only been an indirectly and briefly broached research area in previous research work.

Previous related work focuses on a range of related topics and concepts without clearly defining energy agility. Researchers have worked on making systems more energy proportional with a higher integration of renewable energy. There has been work on server and cpu power states, energy agile networks, designing a representative problem such as sort, leveraging energy storage, designing energy proportional networks, etc which are all very relevant to different aspects of my own work. In this chapter I will discuss such resources and comment on their usefulness and shortcomings. In order to highlight relevance to my own work, I have categorized previous

related work into three categories: Scheduling Policy Design, Energy Proportional Design and Higher Renewable Energy Penetration.

2.1 Scheduling Policy Design

Variable energy is being introduced into the national electric grid at scale. Data centers are also looking to reduce their operational costs and environmental impact by participation in demand response programs. Hence data centers have to deal with possibly fluctuating input power. Realizing this paradigm with backup generators to fill energy demand gaps or grid scale energy storage to smooth the power fluctuations is impractical because of costs, therefore we need to look at alternate mechanisms to deal with such fluctuating power. Data centers typically come equipped with mechanisms which allow for dynamic energy consumption adjustment and deal with deferrable workloads. This allows for developing work scheduling policies where a deferrable work load can be scheduled at a time when ample energy is available.

Andrew Krioukov’s work ”**Design and Evaluation of an Energy Agile Computing Cluster**” is such an example. Although this work uses the term *Energy Agility*, it does not clearly lay down any boundaries of what it means. The work continues to acknowledge energy agility as a distributed systems problem but applies the term to work loads rather than the cluster itself, terming them *Energy Agile Loads* without actually laying down a definition for energy agility. It is interesting to note that the ability to defer a workload or *slack* as the paper terms it is not necessarily present in all kinds of work loads. Where batch jobs may have some slack, even that is exploitable only to a certain extent which does not make a data center completely energy agile.

This paper goes on to emphasize power proportionality amongst the individual nodes of the clusters and the importance of choosing efficient and agile hardware building blocks. However it also accepts that even in the cluster built by the re-

searchers themselves, the servers with intel atom dual core boards at 30% utilization consume 75% of their peak power. Interestingly however the researchers do not go on to point out the power states in the different components of the servers which do not allow for energy proportional design such as the network interface card and disk. This could have been used to motivate component designers to make more energy proportional designs which can allow the servers a greater range of dynamic energy consumption as the workload varies across the components. The paper however recognizes that servers can not be truly power proportional which is why it is important to leverage other power states of the server which consume negligible power and can be switched into with minimum delay. This is why the ACPI S3 state can serve as an important mechanism for a cluster wide energy agile operation. Although this concurs with my work, we also recognize the importance of the time it takes to transition into the ACPI S3 state unlike this paper.

Finally this work also focuses on a very narrow possibility of workloads. Although the workloads used in the simulation and prototype are based on interactive workloads from Wikipedia and batch loads from two large scientific computers including the National Energy Research Scientific Computing Center (NERSC), they ignore a very important aspect of the typical massively parallel distributed applications run on data centers i.e. inter node communication. The scheduling policies developed in this work, although very relevant in the context they are used, would be ineffective when used with workloads which need to communicate with other nodes. Such nodes would be interdependent and need to be up and running together so that they can talk. This paper assumes that communication would only be required amongst a few nodes so that only a small subset of the nodes need to run together. For a massively parallel application such as Map Reduce or Sort, this assumption does not hold. Such applications require all the nodes involved in the work to be up and running together, a problem not resolved through scheduling.

Similar scheduling work in the paper **Scheduling for Reduced CPU Energy** by Mark Weiser also focuses on reducing energy usage in portable computer systems by utilizing the cpu frequency power states and lowering the clock cycles wherever possible. Although this work focuses only on reduced CPU energy through scheduling algorithms, they provide a valuable insight into the possibilities of saving energy or shaping the energy demands of computing systems by simple scheduling algorithms. Today's modern servers come equipped with many such power saving states and features which can be exploited to save energy. A good example are the CPU idle states (C states) and Dynamic Voltage Frequency States (DVFS, P States). This paper exploits P states to gather energy savings for portable computers.

The work however focuses on developing algorithms by optimizing load traces recorded from Unix machines and simulating scheduling algorithms to determine energy changes. As recognized in the paper itself, this work is limited to simulations which of course are not as good as actual implementations. The paper goes on to claim power savings of up to 70% with its scheduling algorithms. Since these claims are only backed by simulations based on certain assumptions and only limited workload traces are simulated, it is far from any kind of practical application. Nonetheless the paper does illustrate its point that exploiting cpu frequency power states through scheduling policies can be very helpful in saving energy and shaping energy demands for computers using its graph and scheduling policies.

Implementing scheduling policies for energy agility or energy efficiency can be magnitudes of times more effective if there is an accurate predictive algorithm available. Such an algorithm can be useful in predicting work arrival or input energy fluctuations and make scheduling decision accordingly. Fortunately, there has been plenty of work regarding predicting energy and work arrivals in data centers which can be used for better renewable energy penetration and reduce costs due to brown energy usage.

One such paper, **Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers** by Baris Aksanli focuses on such prediction models. He correctly recognizes that data centers have longer running batch jobs whose performance is measured in throughput and job completion times such as Map Reduce, Dryad, etc. The paper emphasizes the importance of using green energy and explains how it could be done so with the prediction mechanisms developed in this work. It goes on to claim increases of renewable energy usage by 3 times with a 1.6 times increase in the amount of work performed by green energy over brown energy.

By developing and implementing its energy prediction models and leveraging them through scheduling policies, the paper shows that renewable energy efficiency can be improved by up to 3 times when compared to instantaneous renewable energy usage. Since one of the base algorithms used in this work is Map Reduce which can be divided into subtasks and simply rescheduled in case there is not enough energy, scheduling with instantaneous power requires a lot of subtasks to be rescheduled. Using a prediction model for renewable energy availability however, such rescheduling tasks can be reduced by up to 7.7 times according to this work. This work focuses on wind and solar energy prediction models which are the most widely used variable forms of renewable energy. The work concludes that its prediction models for variable energy sources with volatile weather conditions can be accurate with up to an 8% mean square error. However as with last paper discussed in this section, this work does not deal with applications which may require communication amongst all the nodes in the system to complete the task such as a distributed Sort.

Scheduling policies and prediction models can be useful tools for developing energy agile and energy efficient systems. Despite their shortcomings, all the works discussed in this section illustrate the potential of using the rights policies at the right time. My work involves similar scheduling policies which are focused towards enabling inter

node communication in parallel tasks under variable power conditions. The policies I have designed are a slightly different flavor as they manage hardware power states rather than schedule the work itself.

2.2 Energy Proportional Design

The principle behind energy agility i.e. use as much energy as possible whenever possible can be satisfied by theoretically purely energy proportional systems. However servers are far from being purely energy proportional. Although certain components such as cpu and memory can exhibit energy proportionality, the vast majority of computing subsystems are not energy proportional. As explained in the next section, many mechanisms exploit the ACPI S3 state which provides near zero power consumption for an inactive server which can be quickly activated. This power state can be used to provide energy proportionality for the cluster as a whole by developing policies to put certain servers of the cluster to S3 when possible.

An interesting paper which makes a good case for Energy Proportional computer system design is **The Case for Energy-Proportional Computing** by Luiz Angra Barroso. This paper motivates the design of more energy proportional computer systems by providing some valuable insights. This paper notes that servers are much more inefficient at lower utilization rates, notably energy efficiency drop below 50% when the utilization level decreases under 20%. Consequently, as servers in data centers mostly spend their time between the utilization rates of 10% and 50%, they are not as efficient as they could possibly be. Hence this makes a case for more energy proportional machines.

The paper goes on to say that inactive power states are not very useful for providing energy proportionality at a cluster scale as computer servers may possess data which might be required by other applications even if the server itself is idle, requiring it to stay active. The paper also mentions that clusters typically face small windows

of idle times, and given the high activation and deactivation times of inactive power states, they simply cannot be used. However as seen by other work in the next section, these conclusions are not necessarily true. Intelligent policies can make data available for servers and exploit the slack in tasks to gain the energy savings provided by inactive power states. Furthermore, works such as PowerNap show that extremely fast times for going in and out of inactive power states could be achieved with some modifications to the OS.

This paper goes on to point out that servers already have a very dynamic power range, changes of up to 70% of peak power can be expected depending on the utilization rates of the servers. Further increases in the dynamic range of the power range, of up to 90% that of peak power, could cut the energy used in data centers by one half. This might be true, but achieving this milestone is still far. The components in computer systems that consume more power and are not power proportional are memory and disk storage. Although memory is about 50% energy proportional, disk drives are only 15%. My work in later sections shows that for certain disk types, the energy proportionality is even lower. Even with advances such as SSD in disk storage, energy proportionality still has not improved. The same stands for many other components such as network interfaces and cooling equipment. Therefore we have to look at inactive power states in our work to provide energy proportionality at a larger, more abstract cluster level of operation.

Continuing the discussion of energy proportional components of a computer cluster, Dennis Abts introduces his interesting work in the paper **Energy Proportional Datacenter Networks**. As data center servers become more energy proportional, the proportion of energy consumed by the network will rise according to the study, and may take as much as 50% of the total data center energy. The paper proposes an interesting network topology called the flattened butterfly topology to connect the servers and uses low power state links to design a network which consumes a

simulated 85% less power. The author argues that since data center servers are never fully utilized, using links in their slower, lower power states is justified. The paper then goes on to claim a fantastic cost savings of 2.4 million dollars over a period of 4 years for a reference data center. However considering all the work is based on simulations, it is difficult to take this claim seriously. Nevertheless this work does provide a mechanism to make energy data centers more energy proportional and can be implemented for better power savings.

2.3 Higher Renewable Energy Penetration

One of the basic purpose of my work is to allow deeper renewable energy penetration by allowing data center applications to be more tolerant to input power fluctuations. As such it is important to examine previous work which can help design mechanisms and tools allowing greater tolerance to input power. Previous such work although oblivious of the principle of energy agility i.e. to use as much energy as possible, whenever possible nevertheless provide interesting examples of enabling data center applications to work with renewable energy and input power fluctuations.

One such good example is the work **GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks** by Inigo Goiri. This paper introduces a wrapper built around Hadoop to enable it to allow better renewable penetration; called GreenHadoop for the purpose of the paper. GreenHadoop leverages the ability to schedule Map Reduce jobs within Hadoop to leverage greater use of renewable energy, specifically solar energy. In addition, this paper also introduces solar energy prediction mechanisms based on weather forecasts and cost aware job scheduling to minimize the cost impact of brown energy usage.

The paper starts with the interesting concept of net-metering in the context of renewable energy. Net-metering in essence means that renewable energy users and producers can feed the electric grid with the excess green energy for money. However

inefficient voltage transmission, the unavailability of net-metering everywhere as well as the possibility that retail price for energy may not be offered for net-metered energy make it impractical to use for data centers. Data centers must therefore look towards utilizing green energy whenever it is produced. Storing green energy at large scale in batteries for later use is also not practical as batteries incur energy losses due to internal resistance and self-discharge, battery costs can dominate renewable energy powered systems and because batteries use chemicals which are harmful for the environment.

The paper for this reason introduces GreenHadoop along with Map reduce job scheduling and data management techniques based on the solar energy prediction model developed in this work. GreenHadoop claims increased renewable energy usage by up to 31% and decreased brown energy usage of up to 39%. The exclusion of batteries in the design however limit the work from working on truly fluctuating input power as when input power is not available, GreenHadoop shifts to brown energy which is assumed to be readily available all the time. This means that energy not immediately used, is wasted in this work. Despite this shortcoming, the paper does exploit the possibility of scheduling Map Reduce jobs to better suit cost aware or energy aware needs of the application depending on the underlying scheduling algorithm used.

The energy prediction model developed in this work is not the most accurate. For times of the year where there is less sunlight, the energy prediction errors can average as much as 23.8%. Nevertheless the application GreenHadoop is designed to work despite such prediction errors. The paper provide other contributions as interesting insights on the effect of renewable energy on data centers. As an example, it shows that with higher data center utilization, the environmental benefits of GreenHadoop tend to decrease. Similarly, with a larger fraction of higher priority jobs, it becomes more difficult to leverage the advantages of green energy as the application is forced

to use brown energy to complete its work. Also since GreenHadoop saves energy by putting its server into the ACPI S3 sleep state, some data is temporarily rendered unavailable. This could be a high price to pay for high priority jobs. This motivates the use of a limited energy storage capability for data centers which can be leveraged to avoid using small amounts of brown energy at a high price and to avoid putting high priority data into unavailable states.

A similar work by the same author i.e. Inigo Goiri in **GreenSlot: Scheduling Energy Consumption in Green Datacenters** also deals with scheduling algorithms in light of the availability and preference of using green energy over brown energy in datacenters. However Greenslot only deals with scheduling batch jobs in parallel whenever renewable energy is available. A similar solar energy prediction module is also used with the scheduling algorithm for better results. The prediction module is designed so that it can just as easily be replaced by a wind prediction model if the datacenter was using wind renewable energy. Greenslot is designed after the widely used SLURM scheduler in Linux and based on scientific workloads in medium to small data centers as opposed to large supercomputers.

Greenslot claims greater renewable energy usage, with an estimated increase of 117% and reduction in the usage of brown energy of up to 39%. The findings in this paper mostly concur with those of GreenHadoop, however Greenslot lacks the ability to deal with typical parallel jobs such as Map Reduce that do not come in batches. As with GreenHadoop, renewable energy usage decreases in Greenslot with higher datacenter utilization. It is also worth mentioning that with the absence of any kind of energy storage mechanism, Greenslot wastes any excess renewable energy that cannot be immediately used.

Another good example of researchers indirectly tackling the challenges related to energy agility in the past is the paper **Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy** by the same author i.e. Inigo Goiri.

Goiri once again motivates the need for the work in the paper by mentioning data centers are shifting to renewable energy and the use of intermittent power sources is increasing which leads to a need for data centers to intelligently manage their workloads along with the sources of energy at their disposal. This work proposes a datacenter comprising of a small container, solar panels, a battery bank and a grid tie. It also proposes GreenSwitch, a model-based approach for dynamically scheduling the workload and selecting the source of energy to use. The author addresses the lack of using a battery bank in his previous work through this paper.

Through the use of energy stores such as net-metering and batteries, Parasol can efficiently manage multiple energy sources (renewable energy, battery energy, grid) to provide reduced overall electricity costs to the data centers. Interestingly, the author of this work argues that using batteries to store energy may not be the most efficient way when net-metering is available. This might be true in where the authors work is located (North Carolina), but for many parts of the world, net-metering is simply not an available solution for energy stores or is not as effective. The paper discusses many interesting factors in setting up data centers with renewable energy such as collocating with an existing data center or generating it's own power. It also touches on how increasing efficiencies of solar cells will encourage smaller sized solar farms and reduce the time taken to amortize the capital costs associated with renewables. Hence it concludes that the future seems promising for data centers to invest in their own renewable energy production facilities.

However despite providing valuable insights into how data centers may benefit from a closer integration with renewable energy, this work does not work towards enabling non deferrable workloads to run without the absence of the alternate brown energy. Instead for non deferrable workloads it simply uses its GreenSwitch module to select a suitable energy sources to reduce costs where for deferrable workloads it simply defers the work until a more suitable power source is available. Although this

paper makes for an effective study with claims of electricity cost reduction up to 96% using deferrable loads, it falls short of enabling data centers to be independent of brown energy. It does however help in decreasing dependence on the energy coming from the grid by leveraging it's own solar energy.

The wide range of experiments documented in this paper depict different scenarios of using data centers without any energy stores, using data centers with only battery energy storage and finally using both batteries and net-metering (where the utility pays the wholesale price for energy). Goiri then concludes that by removing the capital costs of batteries associated with data centers and only using net-metering, data centers can amortize their costs much more quickly. This conclusion however is subject to many assumptions including the availability of net-metering. It is also worth mentioning here that many medium to large scale data centers come equipped with energy storage mechanisms such as flywheels to be able to safely power down the hosted servers in cases of emergency outages. These energy storage mechanisms can always be leveraged to provide more renewable energy storage. Nevertheless Parasol is an exemplary green data center for it's time for sizes typical to those found in universities and small enterprises. GreenSwitch is also a good work scheduler when coupled with an energy prediction mechanism, reducing energy footprints to up to 100% in certain cases.

A very important mechanism used in my work is also adapted from a similar work to enable data centers to work on intermittent power sources; namely **Blink: Managing Server Clusters on Intermittent Power**. Blink designs a hardware-software platform with application level modifications allowing servers to be active and inactive periodically for given time periods. This works starts by categorizing work load driven or power driven techniques used in previous literature for data centers working with renewable energy. Work load driven techniques focus on reconfiguring applications as their workload demands vary to use the least amount of energy to

satisfy demand. Power driven techniques on the other hand reconfigure applications as their power supply varies to achieve the best performance possible given power conditions. Blink uses policies to determine a balanced technique taking into account both the workload characteristics and the energy constraints. The goal of this work is to minimize performance degradation as power varies.

Interestingly, without clearly defining energy agility, Blink does satisfy the principle of using as much energy as possible whenever possible to complete a running application. It provides an application independent mechanism to enable data centers to run on intermittent renewable energy. However the mechanism does not exactly provide an energy proportional performance from the data center especially when the time taken to go into the S3 ACPI states used in blink is a large proportion of the blink interval.

Blink is a unique innovative mechanism which can be useful in a number of scenarios for data centers. However the paper does have certain questionable assumptions. It builds its work on the fast S3 transition times mentioned in the PowerNap work. PowerNap claims unreasonable fast S3 transition times based on simulations and disabling of certain important modules which are needed for normal application work e.g. the Network Interface Card. Blink then goes on to implement its mechanism with a low performance cluster based on atom processors with real renewable energy sources of solar and wind power. Although its real implementations are very insightful and show its practicality in real world situations, it does assume the presence of a very large battery storage. Such a battery storage buffer which is capable of powering a normal sized buffer for up to 14 hours is a significant investment which is not present in every data center. Furthermore, large energy storage banks also increase the time it takes to amortize capital costs for data centers, hence departing from the whole principle of the work i.e. reduce costs through higher renewable penetration or increased demand response program participation.

Other than introducing the mechanisms behind blink, the paper further derives blinking policies which can be useful for different scenarios depending on the type of application being used in the data centers. Four policies of Active, Synchronous, Asynchronous and Asymmetric blinking are derived from this work. It might be worthwhile to note though, that the paper only experiments with the memcached application. Although this is a popular application, it does not cover the complexities in other interactive applications such Map Reduce and Distributed Sort which deal with internode dependence. Application performance is also measured as a function of hit rate and fairness rather than throughput and completion time as is more common for interactive applications. Blink does recognize that the policy used is dependent on the type of application, however I find that for most interactive application where internode dependencies are plentiful only the synchronous policy performs well. Nevertheless Blink is an important mechanism for my work. I have implemented the synchronous policy for my application as explained in detail in later sections.

Benchmarks are an important way of establishing baselines to compare different systems. A good example is the Benchmark established by Suzanne Rivoire in the paper **JouleSort: A Balanced Energy-Efficiency Benchmark**. This work first defines it's own sorting system based on commodity hardware and then goes on to set a benchmark to compare the efficiency of different sorting systems. The work notes that to drive improvements in any research area, benchmarks are necessary tools to assess the effectiveness of various solutions. It further defines distributed sorting as a good base application for developing benchmarks as it tests all the basic components of the system such as cpu, memory, I/O, persistent storage, etc. Three categories of sort are defined for fixed input data size and the sorting systems are evaluated based on the minimum joules per record achieved for each category.

This work has been phenomenal in driving energy efficiency work with respect to sorting applications. Such work can then be easily expanded on to other applications

to derive more benefits. It is interesting to note that the authors consider a variety of metrics for measuring energy including setting up a fixed energy budget, a fixed time budget but finally decide on using the fixed input size metric as it provides for an easy measure of the energy spent in trying to sort the input data of three different sizes. This energy per record measured for the benchmark is done at a fixed temperature range of 20-25 degrees celsius including all the devices used to support the sort such as cooling equipment etc. Not surprisingly, the paper shows that using balanced systems for sorting where one or few hardware form a large bottleneck over others e.g. I/O, network do not perform as well as balanced hardware. According to the paper, cpu bound processes are generally more efficient. The paper then defines in detail it's own sorting system called joule sort which performs better according to the defined benchmark than previous existing sort systems.

CHAPTER 3

ENERGY AGILE COMMUNICATION

Energy agility is a relatively new concept but as explained in the previous sections, it has a rising importance. As modern data centers deal with intermittent power sources and participate in demand response programs, they have to address energy agility either directly or indirectly. The alternative to energy agile design for data centers is to wait for an ideal period of stable power or low energy demand to schedule work as explained in some of the previous works. This may mean losing a lot of power, especially when power fluctuations or changes in power demands over the grid are very high. Previous related work shows that some researchers have tackled this problem indirectly by proposing solutions which schedule the available work at suitable periods of power availability, making or motivating more energy proportional computing systems designs which can respond to power fluctuations with a higher dynamic power range (GreenSlot [12], GreenHadoop [14]) or by introducing novel techniques such as Blink [29] which allow data centers to deal with power fluctuations.

Contrary to previous approaches, I use this opportunity to lay down a clear picture of what energy agility is, why it is needed and then build a foundation problem which tackles the issues faced by typical data center applications when running under variable input power. JouleSort [28] is an exemplary work in this regard; having defined the problem of energy efficiency, the researchers from that piece of work continue to build a framework for their benchmark. This framework was later used by a myriad of applications to develop energy efficient systems and help spur on further research in that area.

Datacenter workloads can be categorized into two categories; batch workloads and interactive workloads. Batch workloads are typically scientific or data centric jobs which may have loose timelines and typically do not communicate much with other nodes in the datacenter. As a result they possess a certain *slack*, meaning they can be rescheduled to better suit the power available to the servers. Interactive workloads for data centers on the other hand work with tight timelines or being massively parallel deal with heavy communications with other data center nodes. Common data center applications such as Dryad [?], Distributed Sort, Map Reduce [7], Cassandra [19], H-Base [5], PIG [?], etc tend to be massively parallel requiring communication or a lot of data transfer amongst nodes.

Interestingly, internode communication or data transfer can be a model problem for energy agile systems. As previous work shows, servers can be easily put into active and inactive power states with a small penalty of a time delay when the server performs the necessary tasks to put itself in inactive states. This delay of going into and out of inactive states is called transition time for the purpose of my work. The server can possess many inactive power states, but for my work it suits to work with the ACPI S3 inactive power. S3 consumes negligible power when inactive, but since it preserves memory it allows for a fast transition back into active state where the application can just continue with it's work. Internode communication and data transfers create a dependency amongst nodes where the nodes involved in the task have to be simultaneously active to finish their work. For a task involving a large number of nodes, potentially the whole cluster, this would present a problem when there is not enough power to simultaneously keep all the servers active. Contrary to internode communication, tasks without any internode dependencies can simply be rescheduled to suit power needs for the server. Keeping in mind the ample work on rescheduling task policies in the previous chapter, we can use existing work for this rather than reinvent the wheel. Internode communication therefore is a good representative

problem for an energy agile data center application as it is a parallel data intensive application which forms a subtask of many common data center applications such as FIG,

3.1 Emerging Scenarios

Energy agility can be useful in a number of possible scenarios. Previous work shows that optimizing for power variations in data centers assumes *soft power caps*, so that when using power at a given time becomes undesirable, because it is expensive, "dirty", etc, it is still always available as an alternate option if necessary [13, 14, 35]. Energy agility on the other hand defines a *hard power cap* where there is no alternate option for power. By completely removing dependence on any alternate power sources, hard power cap yields significant cost and carbon reductions.

As an example, data centers can increase their reliance on local intermittent renewable without the use of any expensive storage devices or by relying on the grid for backup power. An increasing number of data centers are now operating off local or nearby large-scale solar and wind farms[10, 25], with the prominent example of Apple's new iCloud data center co-located with a 20MW solar farm [1]. As such facilities look to maximize the benefits from renewable sources and reduce their dependency on energy from the grid and stored energy, they need to look at varying power usage.

As an alternate, there are arguments which emphasize that data centers should not be co-located with renewable sources, leaving renewable integration to the utilities that operate the grid [15]. In such a situation, data centers can still acts as *Load Resources* (LRs) for the grids ancillary service markets [34]. LRs can be thought of as "reverse generators" where the grid controls such LRs to manage power demand to better suit the power supply. This involves signaling the LRs to increase or decrease their power consumption over time according to the market conditions. LRs are then compensated based on their ramp time i.e required time to effect a power change and

capacity i.e. the power range they can control. With increasing renewable penetration, the grid requires more LR capacity, balancing off a volatile demand in power. Since data centers have short ramp times and very high capacities, they make good candidates for LRs.

3.2 All to All Communication

Continuing the discussion from this chapter regarding the need for a good representative problem which addresses challenges applications have to deal with while being energy agile, I introduce All to All Internode Data Shuffling; a simple data intensive application with an emphasis on internode cluster communication. The insight behind this application is the observation that almost all computer applications are built using simple tasks as building blocks. One such elementary task is Sorting. JouleSort [28] defines the importance of sorting in day to day computer applications, in particular the importance of a distributed sort in a multi node data center applications and then goes on to define a benchmark based on a distributed sort with varying input sizes. A distributed consists of several phases which can be completed in different orders depending on the flavor of the sort application being implemented. To explain simply, a distributed sort works on a large chunk of data which has been randomly placed on all the nodes involved in the sort. The data consists of a large file, composed of smaller tuples of 100 bytes each. Each tuple consists of a 10 Byte key and a 90 Byte value. For sorting purposes, the 10 Byte key is equally divided amongst all the participating nodes where at the end of the sort, each node must have all the data which falls within its allotted key range, sorted by the key.

During sort, a copy of the application runs on each node, going through the unsorted data on the node tuple by tuple and identifying which node each tuple belongs to. The application may then create a buffer of a certain size for each node where it stores all the tuples which fall in the key range for that node. Once the

buffer fills up, the application may sort it locally and then send to the destination node where it is stored. Once all the nodes have gone through their unsorted data and sent the buffers to the destination nodes, they must then sort all the chunks of sorted data they received from the other nodes, completing the distributed sort.

Interestingly the phases of sort can be divided into tasks which are being processed locally without internode dependence and tasks which require internode communication (data transfer). As noted before, in case of intermittent power, tasks which can be processed locally may be easily rescheduled based on power availability for the nodes using the rescheduling policies developed in related work. However, dealing with internode data transfers, which require all the nodes to be active for the communication to complete is a different story. With intermitted power, there may not at times be enough power to keep all the nodes concurrently active, making for an interesting problem for energy agile applications. All to All Internode Data Transfer is an application which consists of all of the tasks of a Distributed Sort minus the two sorting operations of sorting the intermediate buffers and sorting the data chunks received at the destination nodes.

Hence all to all data transfer simply means sending the data tuples falling in a specified key range to the node which hosts that key range. A particular insight of this application is that each node has to transfer data or "communicate" to every other node participating in the application (figure 3.1). Since each communication involves two nodes sending data to each other, the total number of communications that must take place for the application to complete is $2 \cdot N C_2$ where N is the total number of nodes. As an example, figure 3.1 shows that for an application involving four nodes, there will be a total of $2 \cdot 4 C_2$ or 12 communications to be completed before the application runs to completion. Each communication involves two nodes sending and receiving data to each other. If we term these two send and receive

communications as a single link, the total number of links required to be completed for the application becomes NC^2 .

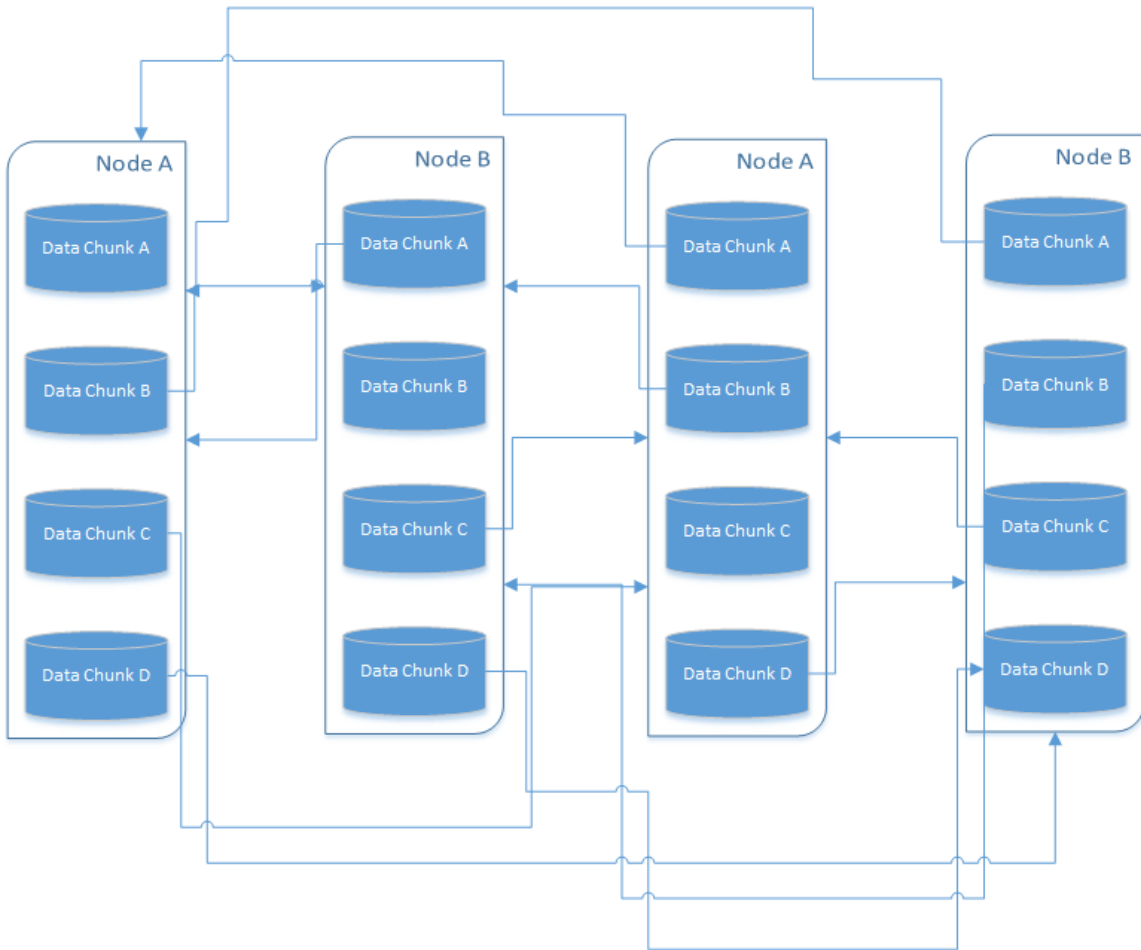


Figure 3.1. All to All Internode Communication

3.3 Application Design

Having defined a problem representation for energy agility in the form of All to All Internode Cluster Communication with intermittent power as explained in the previous section, I go ahead to define a solution in the form of an application to perform this task. This designed solution is implemented in a simulator with the designed environment and in an actual cluster as explained later.

Such an application must complete all links between the nodes to complete all the data transfers. For N nodes, this means completing all NC^2 links with an intermittent power input. Power may fluctuate as the application runs so that at any given time there may or may not be enough power for the nodes to complete their links. If there is not enough power, only a subset of the nodes may be powered up. Hence as the power varies, the subset of nodes selected so that they may complete their links must be intelligently managed in order to complete the work as fast as possible. Application performance, as is the case for most distributed applications can be measured in how quickly it completes. i.e. lower runtime means better performance.

The nodes involved in the application are categorized as manager nodes and worker nodes. Each worker node is given an unsorted chunk of data in the form of a large file. The application is designed similar to a distributed sort, where a copy of the worker application runs on each worker node, going tuple by tuple through the raw unsorted input file and filling up outbound data buffers for each destination node. As the buffers fill up, they are sent to the destination node and the buffer emptied so it could be filled up with more data from the unsorted input file.

At the heart of the application is the manager node which instead of being involved in the data distribution itself, directs and manages the worker nodes involved in the actual data distribution. The manager node is aware of the input power level, the energy storage capabilities available, the current state of each worker node and the number of nodes that can be activated based on the input power. It can also control the worker nodes by putting them to inactive state or waking them into an active state. It is therefore responsible for making the decisions to utilize the available resources i.e. input power, battery storage and worker node states to best complete the All to All Internode Communication.

The design of the application creates an interesting environment which is common to energy agile designs. It is explained in detail below.

3.3.1 Distributed Environment

The aim of this work is to analyze energy agility solutions for data centers. For design, simulation and implementation purposes therefore the target environment is cluster consisting of multiple server nodes connected together through a high speed network. For simplicity and generalization, the application is designed to be agnostic of cluster design details such as network topology, cluster size, power storage mechanisms (flywheels, batteries, etc). As shown in the figure 3.2, we can simply treat the data center as a network of interconnected server nodes (whose capabilities are explained below) with a tie to the power source and an energy buffer which has limited storage capacity. Power source in this setup may be modeled as desired (solar, wind, stable, pulse width modulated, etc) to better evaluate the application design.

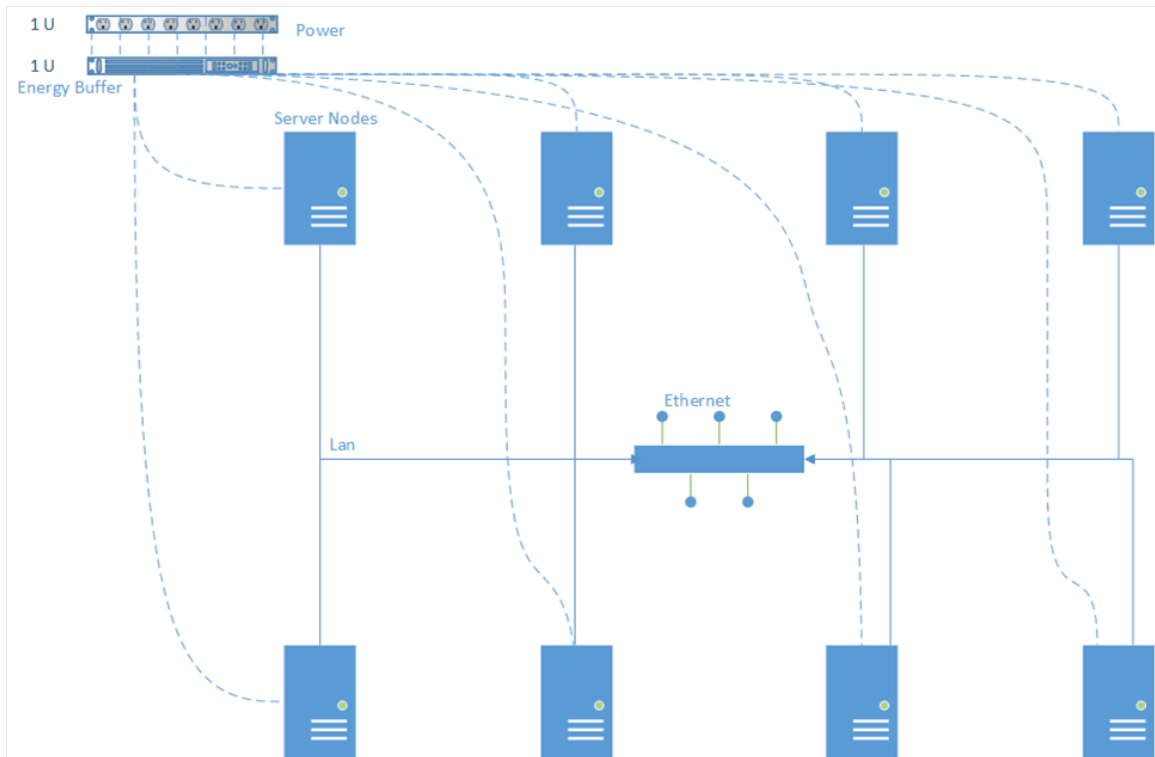


Figure 3.2. Simplified Cluster Diagram

As is commonly found in most servers, our distributed environment assumes nodes to be capable of ACPI power states, specifically ACPI S3. S3 provides fast transi-

tions into an active state where the server consumes negligible power while keeping memory persistent to quickly power back up as explained in previous sections. S3 is a commonly used power state to manage the nodes of data centers as shown in previous research work such as Blink, GreenSlot, GreenHadoop and PowerNap [31, 3, 29]. However, despite our usage of S3, an equivalent power state which allows near zero power consumption in idle state and provides a fast power down and transition up can just as well be used in my work.

High performance servers such as those found in the MGHPCC possess many other "active" power states as well besides ACPI such as the idle C states or the active P states [6, 11, 21]. Alternatively, there are power capping mechanisms using the acpypad driver to enforce a maximum utilization rate in the CPUs by injecting idle states to maintain the average power consumption of the server at a specific level. Similarly CPU cores can simply be disabled, especially in linux where they are hot plugged. These power states help the server to maintain its dynamic power consumption which can also be useful for energy agility, when the variations in input power are manageable by the servers' power usage range. Typically such "active" power capping mechanisms are not useful over a large range of dynamic power range for the servers .

C States: C States are CPU bound mechanisms carried out when the CPU is idle to save power. Various components of the CPU are powered down in different ways to save power. The deeper C state you go into, the more power you save while incurring a higher time penalty of going into the state. Table 3.1 lists various C states, their names, the processors they are available on and the inherent power saving mechanisms carried while going into that state. As of this writing, the highest CPU idle C state being used in modern servers is the C7. According to intel documentation, C7 is similar to C6 with the exception of flushing out L3 memory if all the processors are in C7.

State	Name	Processors	Meaning
C0	Operating	All	CPU 100% turned on
C1	Halt	486DX4 and up	CPU main internal clocks off via software, IC/Bus on
C1E	Advanced Halt	Socket 775 CPUs	CPU main internal clocks off via software, IC/Bus on
C1E	Advanced Halt	Turion 64, 65-nm Athlon X2, Phenom	All CPU internal clocks off
C2	Stop Grant	486DX4 and up	CPU main internal clocks off via software, IC/Bus on
C2	Stop Clock	Pentium, K5, K6...	CPU internal/external clocks off via hardware
C2E	Extended Stop Grant	Core2Duo and up	CPU main internal clocks off via hardware, less voltage, IC/Bus on
C3	Sleep	Pentium II, Athlon and up ex. Core2Duo E4000, E6000	All CPU internal clocks off
C3	Deep Sleep	Pentium II and up ex. Core 2 Duo E4000, E6000; Turion 64	All CPU internal and external clocks off
C3	AltVID	Turion 64	All CPU internal clocks off, reduced voltage
C4	Deeper Sleep	Pentium II and up ex. Core 2 Duo E4000, E6000; Turion 64	CPU voltage reduced
C5	Enhanced Deeper Sleep	Core Solo, Core Duo, 45-nm mobile Core 2 Duo	Voltage reduced like crazy, memory cache turned off
C6	Deep Power Down	Core Solo, Core Duo, 45-nm mobile Core 2 Duo	CPU internal Voltage reduced to anything

Table 3.1. C State Details

P States: Unlike C states, P states are active states which invoke Dynamic Voltage Frequency Scaling (DVFS) to save power while lowering the operating frequency of the processors. The number of P states available in any server or the number of

clock speeds the CPU is capable of operating depends on the processor being used. P states are generally controlled by the operating system through various governors which make decisions based not he server workload. Ubuntu kernel 3.16 uses the governors:*Performance Governor, PowerSave Governor, On-demand Governor and the Conservative Governor* for managing its P States. Additionally the P states can be manually controlled using userspace programs.

Hot Plug CPUs and ACPIPAD: Modern servers support Hot Plugs for several of their components, including CPUs. CPUs can simply be disabled while the server is carrying out its normal workloads. The operating systems deals with the various complexities of carrying on the applications without any disruptions with the other operating processors. Another power save mechanism, present in some of the modern servers using intel processors is the ACPIPAD driver. This driver force injects idle states into the CPU to maintain a certain average power consumption level specified by the user. This is part of Intel’s PowerClamp mechanism to achieve forced and controllable C state residency to control a system’s power consumption.

3.3.2 Policies

Following up the simplistic environment design explained in the last few sections, a manager node is designed to be able to control worker node power states, putting them to active or inactive depending on the available input power and energy storage. The manager decisions follow an algorithmic approach based on different policies. It pools the energy sources and decides which nodes are the best candidates to keep activated, deactivating the rest. To complete the work, it must manage the nodes the most efficient way so that all NC2 links of data are completely transferred. Given the possibility however that only a subset of the nodes may be active at any given time i.e k, only KC2 links are being worked on for that subset of nodes. At the end of the application, the summation of all the completed subsets i.e. KC2 must be equal to

NC2 for all the links to complete. The four policies designed to achieve this in this work are named: *Active PowerCap*, *Blink*, *Burst* and *Battery Burst*. All of them are explained in the coming sections.

Interestingly, the "policies" mentioned in this work are starkly contrasting with the "policies" developed in previous related work. Where previous work used policies to schedule the tasks at the best possible time, my work instead uses policies to manage the worker node states while leaving the details of how the tasks are managed to the worker nodes themselves.

3.3.3 Design Concepts

The All to All Internode Communication, although a simple application, is designed to be a significant challenge for energy agile systems because it is communication intensive. It consists of some concepts which may be useful in designing a more optimal solution, however are not apparent at first sight. The discussion in this section is in the context of the All to All Internode Communication problem only. In particular, the concept of *balance* can be important for understanding the full breadth of the problem.

Balance When All to All Internode Communication is carried out without hard power power caps, every node simply sends the data to every other node as explained in Application Design. For every node taking part in the application, it possesses a certain amount of data that it must send out to every other node. It must also similarly receive a certain amount of data from all the other nodes in the application. These communications amongst the nodes have been previously termed as "links" in this chapter. A way to imagine the problem is that each node must establish a link with every other node and complete those links (NC2) to complete the application. While running without power constraints, each worker node (all nodes are assumed to be homogenous) is active and sending/receiving data at a speed determined by

the bottleneck of either the network or the I/O. Hence each node is completing their links at the same rate.

With intermittent power on the other hand. The worker nodes will complete their links as directed by the policies deployed at the manager node. These policies may or may not allow all worker nodes to complete their links at the same rate, leading to an unbalanced communication state i.e. some nodes have completed more of their links than other nodes. The downside of having an unbalanced communication is wasted power towards the end of the application. As the nodes are completing their links at different rates, towards the end some nodes will have completed some of their links with the other active nodes but not all. As a result, some nodes will be active together but not communicating, and hence wasting some power. The figure below shows two situations where the application maintains balance in the nodes on the left. The right side of the same figure shows a contrasting situation where the application does not maintain balance.

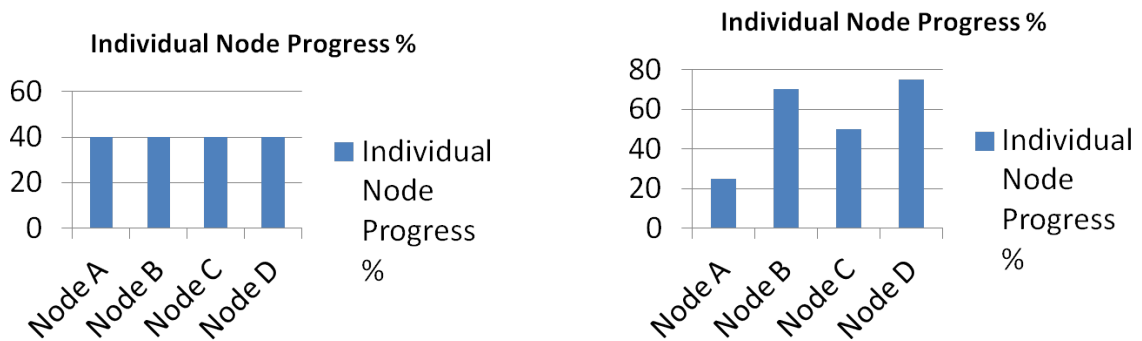


Figure 3.3. Balanced vs Unbalanced Links

Unbalanced communication can also form bottlenecks for application completion in case power suddenly goes full throttle. The application runtime will always be limited by the least completed link since all the links will work in parallel. Such a situation is shown in the figure below.

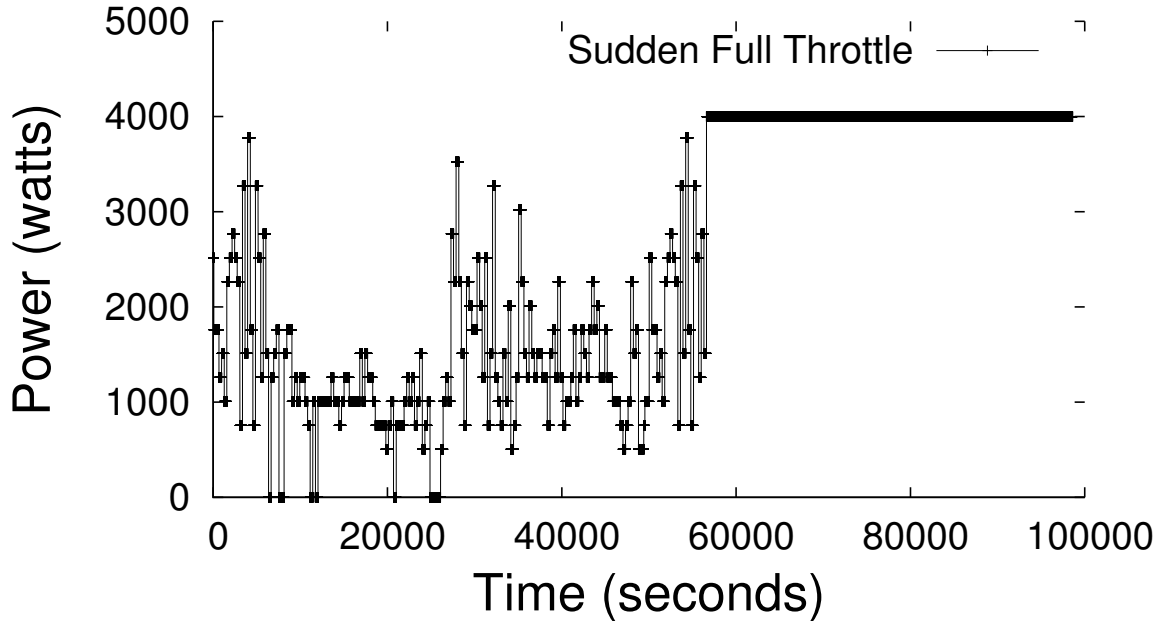


Figure 3.4. Sudden full throttle power

3.4 Energy Agile Policies

This section explains the policies developed and used in my work for energy agility. The algorithm behind each policy is explained followed by the practical conditions where it is expected to perform the best. Available power is assumed to vary every short interval, e.g. 2 minutes.

3.4.1 Active Power Cap

Active Power Capping is simply managing individual worker nodes power caps to match the current power average. Modern data center servers have mechanisms of controlling server power caps through DVFS, C State throttling, etc. Protocols such as IPMI expose interfaces which can be used to manage server power caps easily. The active power cap policy divides the input power to the data center equally amongst all the worker nodes. The sum of each individual worker node equal the total input

power. Active power cap can work without energy storage and maintains balanced internode communication by making sure each node has the same input power.

Active power capping can only be used if the power variations can be dealt with within the dynamic operating power range of the worker nodes. It is more suitable for applications with a lower CPU utilization. Servers may have large dynamic power ranges, however capping them below the power average they use while running the application under normal conditions severely affects the application. Since CPU usually contributes to the dynamic power range with I/O and network not being very energy proportional, the CPU takes a hit when the power is lowered even a little below the application average. This performance degradation can reflect severely in the application performance.

As an example the servers at MGHPCC operate at 215W when fully utilized and 90W when inactive. They also use a network interface and storage which is not energy proportional and consumes 30W each when utilized even a little. Hence with negligible CPU utilization and moderate NIC and I/O usage, the MGHPCC servers operate at 150W. Since my application consumes a little CPU as well, worker nodes when in full swing consume around 165W. Lowering the power cap below this value, severely degrades All to All communication with respect to completion time. In general, we show that active power capping has very little potential for energy agile applications.

3.4.2 Burst

A straightforward approach to inactive power capping is to activate and deactivate a subset of servers based on the available power. The available power is used to activate servers such that "wasted power" is minimized. Power is considered wasted if it is either below a server's minimum active power or above its maximum power necessary to run an uncapped worker. In the former case, the server has only enough

power to turn on and can do no useful work, while, in the latter case, the server cannot make use of any power above some maximum value.

After determining how many servers to activate, the policy must then determine which servers to activate. The servers are selected such that once a server has been activated, it should be kept active until it completes all its links with the current subset of nodes activate. Whenever there the available power increases to that more servers can be selected, new servers must be selected and activated. The server selection is prioritized based on the amount of work they have completed with the other currently active servers. In case every server has completed equal work, the servers can be prioritized from 1...N, such that the highest priority server not yet completed is selected next. Similarly in case power decreases so that less servers can now be supported on the available power, the servers with the least amount of completed links are deactivated.

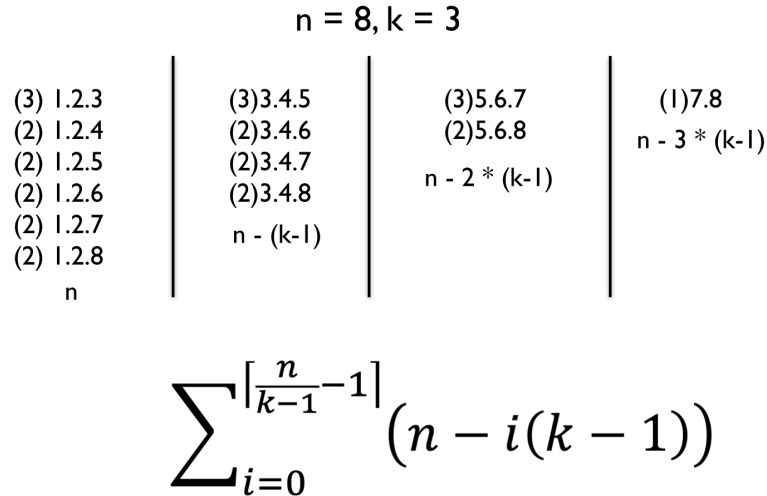


Figure 3.5. Burst Transitions

The burst policy minimizes overhead to transition servers to the inactive power state, since it only transitions a a server to an inactive power state if i) power increases or decreases, ii) a server finishes the first phase and has no more work to do. In fact,

assuming only constant power to activate k servers, the number of transitions is determined by a closed form formula as determined by figure 3.5.

The figure shows an example situation where the total number of nodes n is 8 and the number of servers that can be supported on the current stable power is 3. It shows how the combinations of the servers are cycled through to complete the communication. This pattern of communication is generic for any value of n and k . The first $n-1$ nodes are kept fixed, changing the last node such that all the combinations possible for the $n-1$ nodes are exhausted. All combinations of the $n-1$ nodes are cycled in this, exhausting all possible node pair links while completing the communication. As shown in figure 3.5, while cycling through the number of possible combinations, the number of transitions the nodes go through in the first column is n , the second column is $n-(k-1)$, the third column is $n-2*(k-1)$ and so on until the number of rows in each column is less than equal to 0. The number of rows in the column are determined by the total number of nodes divided by the number of fixed nodes in each column or $n/(k-1)$. The number of transitions are then summed up across the columns to find the number of transitions to complete the application. This summation is shown by the formula shown in figure 3.5.

For variable power however where the variations in power can be stochastic. Whenever power varies, we are forced to incur a transition. Therefore the closed form formula for stable power does not hold for variable power. Instead the number of transitions in variable power is a function of the total number of power variations and the number of nodes in the application.

3.4.3 Blink

The blinking policy is derived from the previous work i.e **Blink: Managing Server Clusters on Intermittent Power**. Over a given time interval, depending on the amount of energy storage available, blinking divides the interval into active and

inactive periods. As an example, if the average power cap per server is 100W and the average power consumption of the application on the server is 200W, the inactive and active periods will be 1 minute each i.e. 50% (duty cycle) given the total interval is 2 minutes. Server latency of waking up and going into the inactive state is part of the active period (overhead). Blinking maintains balanced communication amongst the nodes. The node transitions every interval are controlled by the manager.

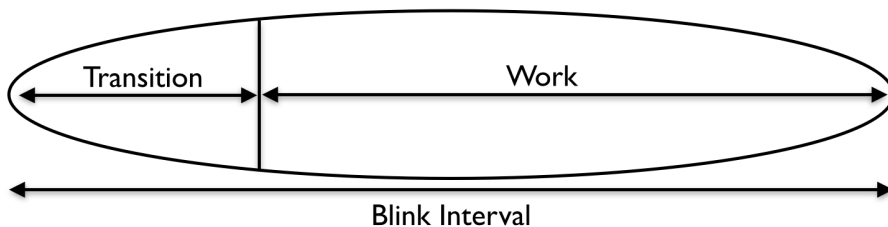


Figure 3.6. Blink Interval

The figure 3.6 shows that the length of a blink interval depends on the size of the battery available. If a large battery is available, all the nodes can be supported on the battery for a longer time until it expires and vice versa. However part of the time of the blink interval is consumed by the transition time in which the nodes are woken up from their inactive state and put back in it. This time is an overhead as no productive work is done here.

Blinking is not affected by the power signals variability as the energy buffer is used to smooth the power supply to the nodes every interval. Rather blinking depends more on the average power delivered despite the power variations. However blinking does incur the overhead of transition time every interval when the nodes go from active to inactive power state and vice versa. If the transition time (dependent on the platform) is very high, then blinking can be an expensive policy.

3.4.4 Battery Burst

The battery burst policy is derived from burst with an additional use of energy storage to minimize the "wasted" power as defined above. This policy stores the excess power which cannot be used to productively turn a server on ("wasted power") into energy storage. When the energy storage is completely filled, the active server nodes are kept running on available power, while the inactive nodes are activated using the energy storage so that all nodes are tuned on similar to blink until the all the available energy stored is used up. Hence this policy is a hybrid between blink and burst.

Interestingly since battery burst is a hybrid between burst and blink, it will perform more like burst when there is not a lot of wasted energy and a very large battery. On the other hand the policy will perform more like blink when the battery is small and there is ample wasted energy such that the battery gets charged more quickly and the algorithm cycles through blink more frequently.

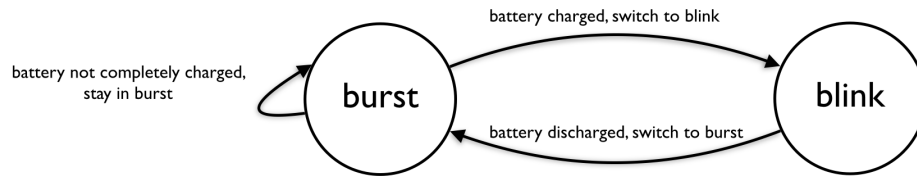


Figure 3.7. Battery Burst

Similar to burst, this policy reduces the transition time overhead by prioritizing the servers. However by using energy storage to activate all nodes when possible, it performs slightly better than burst. It should be noted however that since this policy exploits wasted power, it's benefits over burst are dependent on how much power is wasted.

3.5 Power Input

As energy agility is independent of power signals and their fluctuations, we try to deal with realistic power signals which applications are exposed to today. For this particular experiment, power input is controlled by the manager node which also controls all worker nodes based on the policy being implemented and the current available power. The manager node in my application reads a power file which specifies the power available for each interval. Since energy agility is designed to allow applications to work with renewable power sources and demand response programs, many different power signals adapted from real traces of wind, solar, etc are designed for the application to run on. A few examples are shown in the figure below. These power graphs were generated using real weather data.

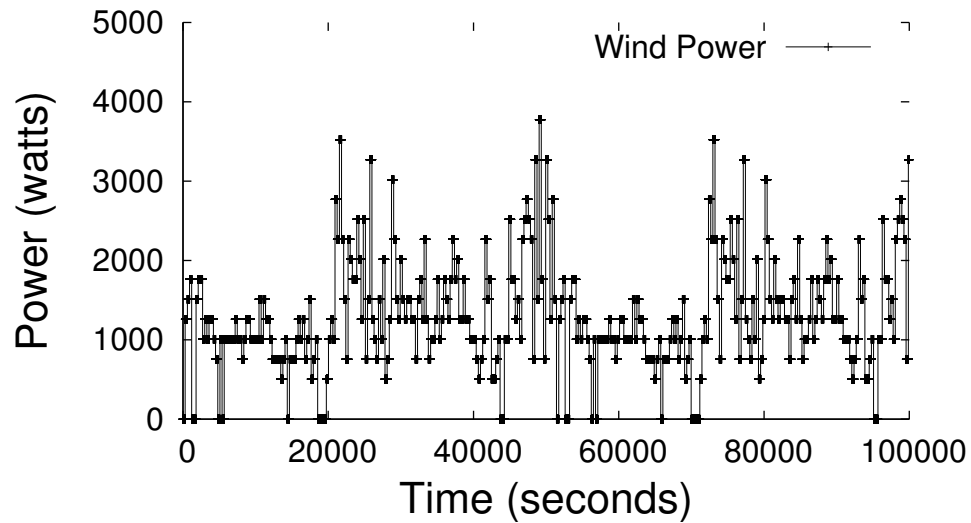


Figure 3.8. Wind Power Signal

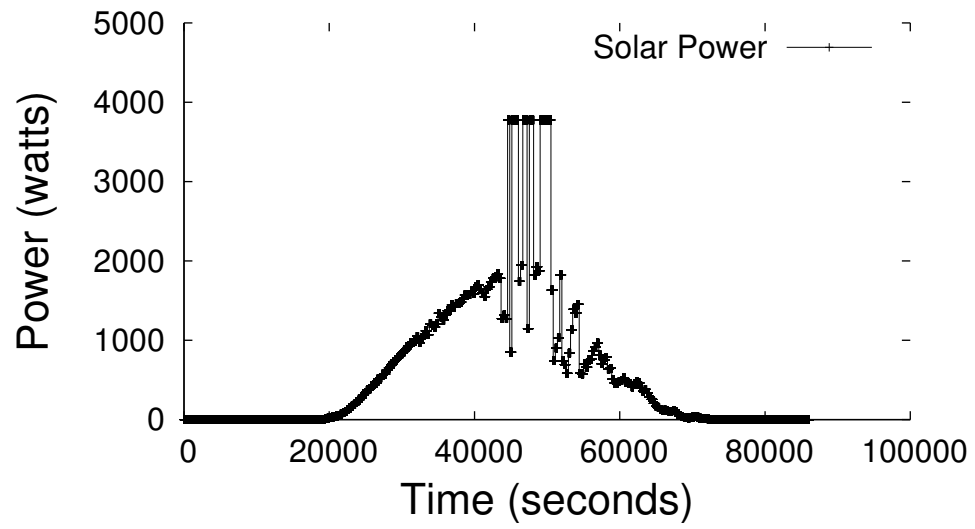


Figure 3.9. Solar Power Signal

CHAPTER 4

IMPLEMENTATION

Following the design and implementation of the model problem, the designs are now realized with a formal implementation. An implementation is of course necessary to validate all the theoretical work and show an energy agile application at work. In this chapter we first talk about a simulator which was used to create a virtual environment where the policies could be developed and the longer experiments carried out much more quickly. Following the discussion on the simulator, the application development and deployment is then introduced and discussed. This is followed by the hardware power state profiling which supports the simplification decisions made while modeling this problem. Many of these graphs provide insight into how clusters can support energy agility. This chapter talks about the various design space constraints and decisions which can impact application development and performance.

The simulator was our first step towards developing the energy agile policies. Implementing the simulations gave an insight into the actual environment and helped with the actual implementations in many ways.

4.1 Theoretical Verification Through Controlled Simulation

A simulator is an imitation of the real world process, in this case an application deployed on a data center. As with every other simulation, it requires that a model be first developed which captures the key characteristics and behaviors of the process and its components. Previous related work shows that simulators can serve to provide valuable insights to the problem by providing a controlled environment for the

problem. It can be particularly useful if the actual implementations of the experiment are expensive, either with respect to time or cost. In the case of a simulation involving data centric massively parallel "big" data applications, this is very true. A single experiment involving 1TB of data may take up to a few days to complete depending on the input power signal and deployed policy. It will also fully employ a large number of data center nodes which can otherwise be used for other productive work. Hence a simulator for energy agile All to All internode communication can save a lot of time and cost in our development and testing. The actual evaluation is then carried out with an implementation on the MGHPCC cluster.

The envisioned use of a simulator in this case is as a *Policy Development Testbed* and a *Policy Evaluation Tool*. The need of both these tools is highlighted below.

4.1.1 Policy Development Testbed

Policies are developed following an idea which may or may not be practically feasible to be deployed on an actual implementation. To implement the theory behind the idea and realize its shortcomings and developmental challenges, it is necessary to have a testbed where the policy can be deployed, tested and debugged before being released. This is especially true if the actual implementations can be costly and time consuming, as in the case of data centric applications.

The simulator developed in this work provides the useful functionality of a testbed. Having developed the model environment described in chapter 3, the environment can also be controlled to test and deploy the policy under different conditions. Additionally, the simulator is designed so that policies can be easily plugged in and deployed on the manager. Since the actual implementation is based on the same language i.e. Java, the policies developed for the simulator can also be simply imported in and plugged into the actual implementation without much more work. Hence the simulator makes an effective testbed because of the ease of exporting the developed policies

into the actual implementation and the realistic environment model created for the policies to be deployed on.

4.1.2 Policy Evaluation Tool

Different policies are designed to work well for different conditions. As an example, blink performs well with a highly fluctuating input power but with low transition times. On the other hand, burst performs well when there is not significant power fluctuations but the transition times are higher. Similarly burst battery capitalizes on wasted power and uses that power to productively so work by leveraging a small energy storage availability. Other than the policies developed in this work, there could be policies from future work which may be designed for other conditions. Actual implementations to compare these policies under various conditions can be expensive, so the simulator can be a valuable alternative.

The simulator can be specially useful for decision making. An application can use the simulator to evaluate the which policy would perform optimally in the context of current environmental conditions such as power variations, transition time, energy storage availability etc. A simulator however is not aware of many real world constraints and conditions, so a simulation follows a more ideal scenario than an actual implementation and may need tuning to match the results of an actual implementation.

4.1.3 Simulator Design

The simulator is designed to model the environment and the problem described in chapter 3. The simulator itself is developed in Java, but is scripted using Python via the py4j library. It can churn out simulation results for varying conditions and plotting out the results using GNU plot.

The simulator simplifies the environment by making generalizations about the object. For example, the servers are treated as nodes with only two power states

i.e. active and inactive. They do productive work when active and consume 165 W power (average power of application running on MGHPCC servers), but are not working or consuming any power in the inactive state. A time latency is associated with each transition into and out of the states, when the nodes consume power but do not productive work. As in the real environment, the nodes are connected through a network, work with some data allotted to them and have an energy storage available. Network speed, amount of data, amount of energy storage amongst other things are all controllable properties of the environment for the simulator.

The manager node owns various policies for energy agility and controls the hardware state of the other nodes. Depending on the policy chosen at runtime, it will make activation, deactivation decision as specified by the policy. It is also made to be aware of the power available to the system. This is done by an input file, containing the power values for fixed intervals. The manager reads the file line by line to determine the power available for each interval. As mentioned in the section Power Input, Chapter 3, the power files can be a wide array of possibilities, ranging from power derived from real traces of wind and solar data to randomly fluctuating traces designed to test the applications energy agility.

The simulator makes many simplifying assumptions and is designed to run for more idealistic conditions. Simulated results therefore show much faster runtimes for experiments than actual implementations. However both the results should follow the similar patterns.

4.1.4 Simulator Output

The simulator was used in my work to generate expected results of experiments before conducting them with the real implementation. The experiments, as explained below, included variations in the environment conditions to analyze their effect on the outcome. As expected, the outputs of the simulator are not accurate reflections of the

real implementations. Nevertheless, the simulator is successful in somewhat correctly reflecting the effects of changes in environment conditions on the experiment outputs. The patterns shown in the simulator are also shown in the actual implementation experiments in chapter 5.

Since my work revolves around energy agile designs, the first simulated experiment was designed to investigate the performance of the various energy policies designed on different energy budgets. Recalling energy agility as the amount of computations done per joule available, a varying energy budget can give us insights into the relative performance of each policy design. The policies implemented in the simulator are burst, batteryburst and blink. Active Power Capping was not implemented as it is dependent on the servers own mechanisms of capping individual server power. Figure 4.1 shows the results generated by the simulator. The experiment was simulated for 500GB data distributed amongst 5 nodes. The x axis of the graph shows the power budget as it is increased from 330W to 825W while the y axis shows the completion time of the experiment. Power budget shows the average power available for to the cluster throughout the experiment. The transition time for the nodes is kept constant at 5 seconds and an energy buffer of 200 KJ is available for the cluster to use.

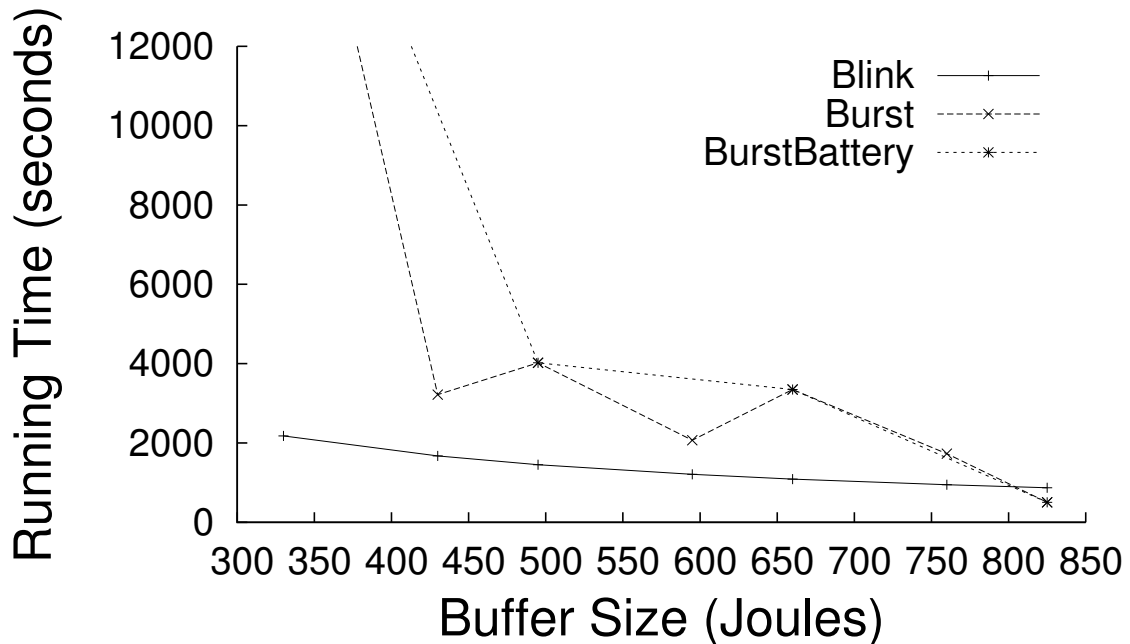


Figure 4.1. Simulations with different power budgets

The simulation shows that as the power budget decreases, the performance of all the policies gets degraded. However the degradation effects are more pronounced in the two variants of burst than in blink. Towards the lower power budgets, blink is shown to perform the best. Burst and batteryburst's performance also decreases as the power budget is decreased. However since batteryburst capitalized on wasted power, it's performance is better than burst, when there is ample wasted power to charge the buffer and use it productively for the experiment. Therefore between points where the power budget can be fully used to power a certain number of nodes without wastage, batteryburst performs better than burst. When compared with the actual implementation however, the simulation shows that blink does not perform as well when the power budget is near full power, which is not true. The simulations also show a more pronounced advantage of using an energy buffer than was shown by the actual experiments.

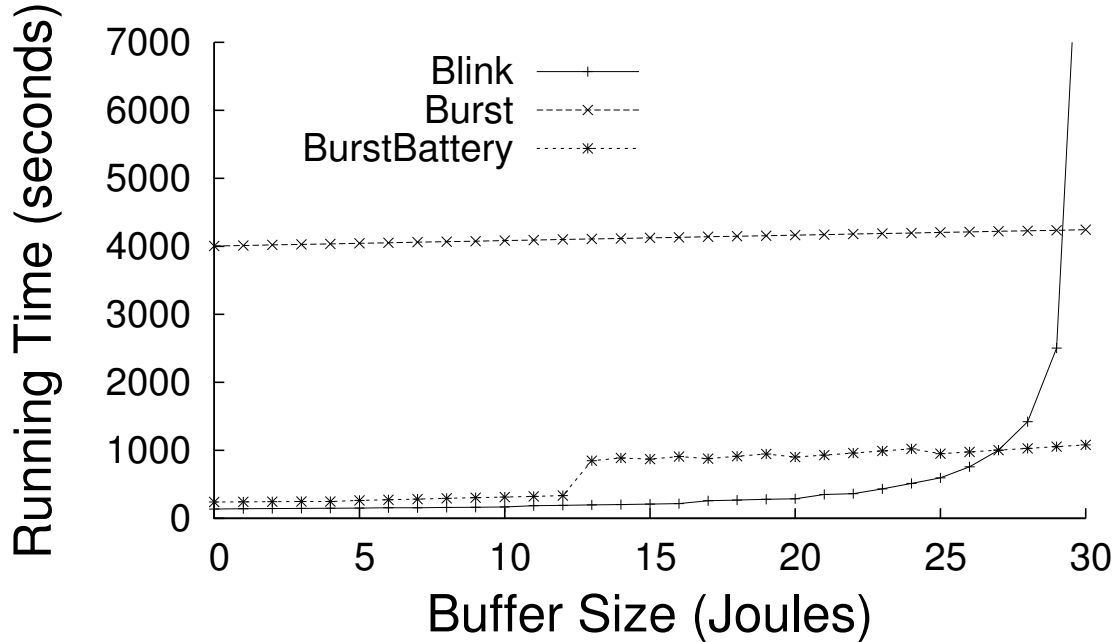


Figure 4.2. Simulations with different transition times

Although energy budget is the most direct measure of energy agile performance, several other factors impact the performance of the various policies. In particular, since burst tries to minimize the total number of transition to complete the experiment, transition time plays an important part in determining the relative performance of the policies. If transition time is expensive, burst and battery burst should perform better whereas for lower transition times blink would be able to do more work in each blink cycle and perform better. Figure 4.2 shows the results of simulations with again the same conditions as mentioned in the start of the section, but with varying transition times of the nodes. Transition time is varied from 0 to 30 seconds as shown on the x axis while the completion time is analyzed for application performance on the y axis. The graph shows that burst and battery burst are not affected a lot by transition time variation in the nodes, however blink performance shows an exponential increase in completion time. These patterns are again confirmed in the implementations shown in chapter 5.

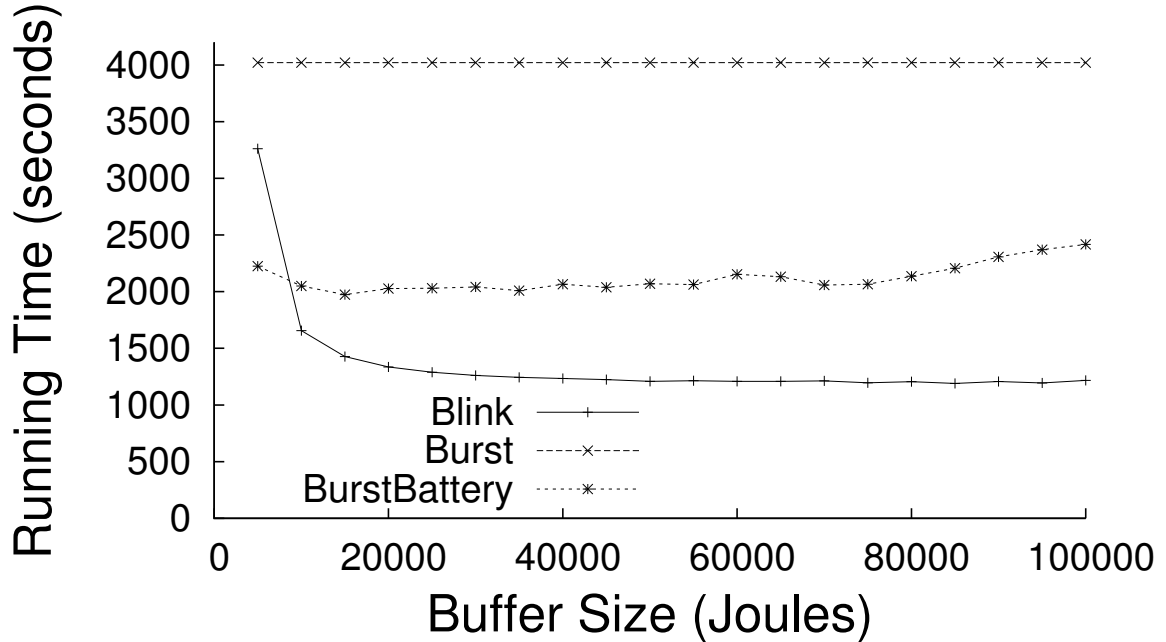


Figure 4.3. Simulations with different buffer size

The major difference between the burst and battery burst is just the use of a battery to capitalize on wasted power. However blink also employs a battery to blink its nodes. It makes sense therefore to compare the relative performance of blink and battery burst for a varying energy buffer size as these are the only two policies which employ energy storage. Similarly, 4.3 shows the experiment simulation for the same environment conditions as before, but for a changing energy buffer capacity. Power is fixed at 595W for this this experiment and the energy buffer increased from 5KJ to 100KJ. The simulations show that burst is not affected by the buffer size, as it works without using any energy buffer. However blink’s performance improves up to a certain limit as the buffer capacity increases while batteryburst shows a not so deterministic pattern. The reason for this is, as the battery capacity increases, it takes a longer time to charge it, especially if the wasted power is less. Therefore at times the application completes before the buffer is charged and can be used, reducing the application time. While at other times, the buffer cycles are just enough to end

just as the application completes, leading to the non deterministic graph shown. This trend is also confirmed by the actual implementation in section 5.

4.2 Application Development And Deployment

The simulator although works with a virtual environment, it has the same environment and virtual mechanisms as those in the real implementation. Hence both the simulator and the actual application have the basics in common amongst them. In particular, they both work with the All to All Internode communication application, which includes one worker process per server and a centralized power manager, both written in java. The workers co-ordinate to distribute the input data while the power manager implements the power management policies from the previous chapter. The details of the worker and manager application along with the platform are discussed below.

Workers. A worker will typically go through the random data file provided to the application serially and build memory data buffers where it stores the data outbound to each node. If that node is active, the buffer when completed is sent to it otherwise the buffer is stored locally on the disk. This continues until the whole data file has been read. The implementation model is adapted from that of Triton Sort. Since the workers are focused to be more energy agile than energy efficient, they are not optimized for the highest possible energy efficiency on our platform. They are implemented with functionality allowing them to pause and resume their work and to report progress with respect to the other workers. The workers are also additionally furnished with necessary functions such as saving a filled buffer into storage instead of sending it if the target node for that particular buffer is inactive.

Manager. The power manager monitors both the servers and the available power available every interval. It then caps the average server power by either capping the power of the individual servers or by activating/deactivating them. In order to

deactivate servers, the power manager first informs all the servers that it is going to deactivate certain servers. The remaining active workers continue to cleanly finish sending any outstanding buffers to the deactivating servers, while the soon to be inactive servers cleanly finish sending the outstanding buffers to all the other workers. Having finished their work, the workers send a notification informing the manager at which point the server deactivates the server. A similar two step process is involved in activating the servers.

Platform. My experimental platform is based on a set of isolated servers in the MGHPCC. MGHPCC is data center dedicated to research computing located at Mount Holyoke, MA. My isolated servers are a set of Dell PowerEdge R720 servers, each with 32 2Ghz cores, 64 GB memory and a 4TB disk. As a result of combining 32 cores with a single local disk, the platform is not very energy efficient for data intensive applications. Because of these inefficiencies, my evaluation primarily focuses on the relative performance of the various policies rather than the absolute energy agility of my platform.

Massachusetts Green High Performance Computing Center



Figure 4.4. Massachusetts Green High Performance Computing Center

4.3 Hardware Power States Profiling

Working with energy agile designs with the worker, manager application we deal with fluctuating input power, exercising the hardware platforms dynamic power range. Hence, it makes sense to explore our platform's (Dell PowerEdge R720) power capping features which allow it to respond to fluctuating input power. The major components of the servers which consume significant power are the CPU, disk storage and the NIC. As such their power states are explored in this section. Working with these states I conclude that the ACPI S3 power state provides a greater dynamic power range which is more useful for a more variable power signal. Active power states are good for handling a smaller variation in dynamic power consumption, however since we deal with stochastic variations in input power it makes sense to have mechanism which can deal with a greater range of power fluctuations.

4.3.1 C States

As explained in chapter 3, hardware capabilities section, C states are CPU idle states. The details of what each C state does and what processor supports it are documented in table 3.1. This section evaluates the C states found in my platform of implementation i.e. Dell PowerEdge R720 servers and the energy consumed by each C state. C states are also important for controlling the processors power cap using the intel power clamp driver. Powerclamp force injects C idle states into the CPU for energy savings, while maintaining the specified average power consumption. Figure 4.5 shows the energy consumed by each C state on the y axis when the CPU is forced to sit idle in it (x axis). All other components are assumed to be idle. It is interesting to note that the CPUs do not possess all C states listed in table 3.1, but they nevertheless have the more advanced C States to invoke higher power savings when needed. The figure shows that for deeper C states, there are higher power savings

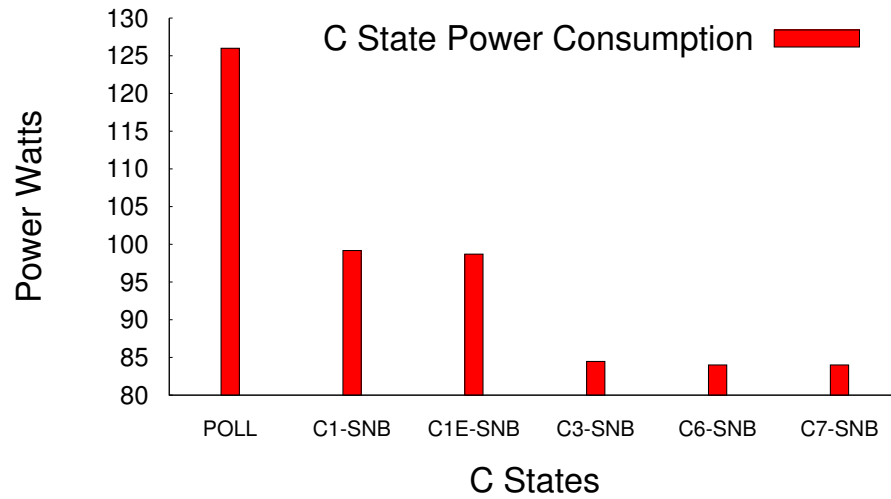


Figure 4.5. C States Energy Consumption

From a system point of view, it is important to note that a processor cannot be forced into a particular C state. A particular C state is achieved by disabling all the C states in the system and forcing the system to sit idle for a while.

The power consumption shown in figure 4.5 is of the whole server, while all it's processors sit in that particular C State rather than just the power of the CPU. Figure 4.6 similarly shows the latency in micro seconds (y-axis) of going into that a particular C state (x-axis) while the necessary tasks of going into that C state are performed. The figure shows that despite the higher power savings in the deeper C states, there is an additional cost of higher latency.

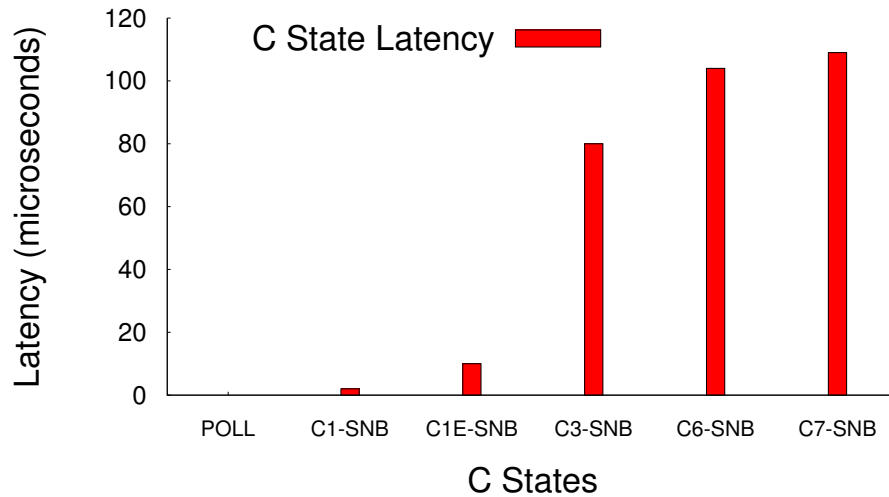


Figure 4.6. C States Latency

When C states are invoked, the CPU sits idle without doing any active work. For applications where the CPU is not completely utilized, C states can be invoked when the CPU is idle to get power savings. The dynamic power consumption of servers can also be controlled by force injecting idle C states to the CPU. The CPU can also control its power states by lowering its frequency and voltage using P states.

4.3.2 P States

P States are active states of the CPU. They employ DVFS to lower the voltage and frequency while saving cubic power at the cost of a lower clock speed. Given that data centric applications are more data intensive with a lower focus on CPU, the lowest P state (in this case 1.2 GHz) can be used for more efficient operation. However despite using the lowest frequency, the CPU for data center applications may still be utilized less than 100%, as is the case in my application, allowing for the use of C states in conjecture with the P states for even more power savings.

Figure 4.7 shows 5 of the 10 P states available in a Dell PowerEdge R720 server. The figure shows CPU utilization against the power consumed by the server in dif-

ferent P states. A P state can be invoked by the userspace governor. The different P states show significant power differences at higher CPU utilization rates. The significant difference between in the power of the highest and second highest P state is due to over clocking (Highest P state is Turbo mode). P state latency is a fixed 10 nano seconds for any P state.

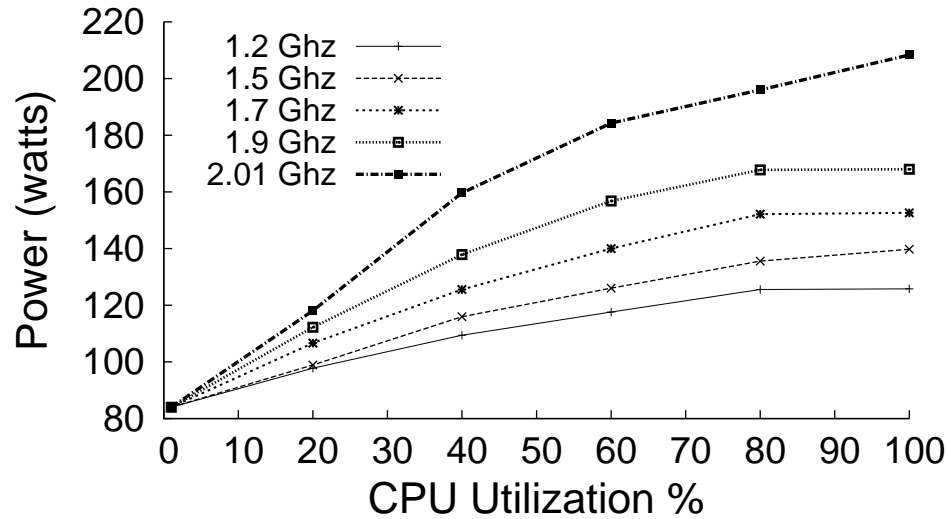


Figure 4.7. P States Energy Consumption

P states are useful in saving power without putting the CPU in idle state. They are specially useful when the CPU utilization is low. There are several governors available in the linux 3.15 kernel which manage CPU P states according to various policies. However for manually setting a P state to control the dynamic power range of servers I use the **User Space** governor.

4.3.3 Controlling CPU Cores

Another possible mechanism for active power capping is disabling CPU cores altogether to save power. I explore the possible power savings from this mechanism in this section.

Following kernel 2.4 (January 2001), hot plugging has been a standard feature of linux systems for a while. It now includes hot plugging CPUs as well such that they can be plugged in and immediately used, using the configuration details to the OS itself. Consequently processors can be just easily removed or hot plugged back into the system to enable power savings. Figure 4.8 shows the number of CPUs on the x-axis and their power usage on the y-axis at different levels of utilization. All the CPUs are individually utilized at the specified levels. The figure shows linear increases in power usage as the number of hot plugged processors increase from 1 to 32. Interestingly however, no matter how many processors are hot plugged at zero utilization, the power consumed is always at the minimum power level for the server i.e 85W.

This graph serves to show two points. The first being that C states are effective at power savings as at zero utilization, even when all the 32 processors are active the server consumes the same power as when there was a single unutilized processor in the system. When inactive for a long period of time, the servers are put into deep sleep C7 state where the power consumed by the processors is almost zero. Secondly, because the C states are so effective in power savings, it does not make sense to use hot plug mechanisms for power savings in the servers, either for energy agility or energy efficiency.

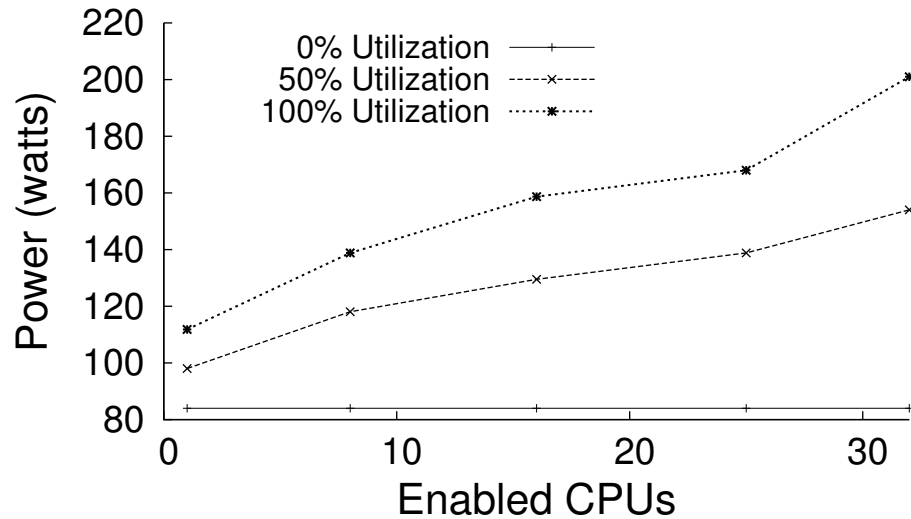


Figure 4.8. Effect of CPU Cores on Power

4.3.4 Throttling Disk IO and Network Bandwidth

The previous section shows the various power states for processors and how they contribute to the dynamic power range of a server. I explain the power states of the other two major power consuming components of a server i.e. I/O and Network here and motivate the energy proportional designs for more computer components. In the absence of a larger operating dynamic power range of servers, it makes sense to work with ACPI S3 states to provide energy agility for largely fluctuating power signals. As shown in later figure 5.3, active power cap takes a large hit when the power usage of server is made to fall below the application average.

Figure 4.9 shows the power usage (y-axis) of the network card and the storage disk as they are throttled at different utilization rates (x-axis). The network bandwidth was throttled between two servers by sharing a file using IPERF, while the I/O disk bandwidth was throttled by transferring a large file from one place on the disk to another at different utilization rates using Rsync.

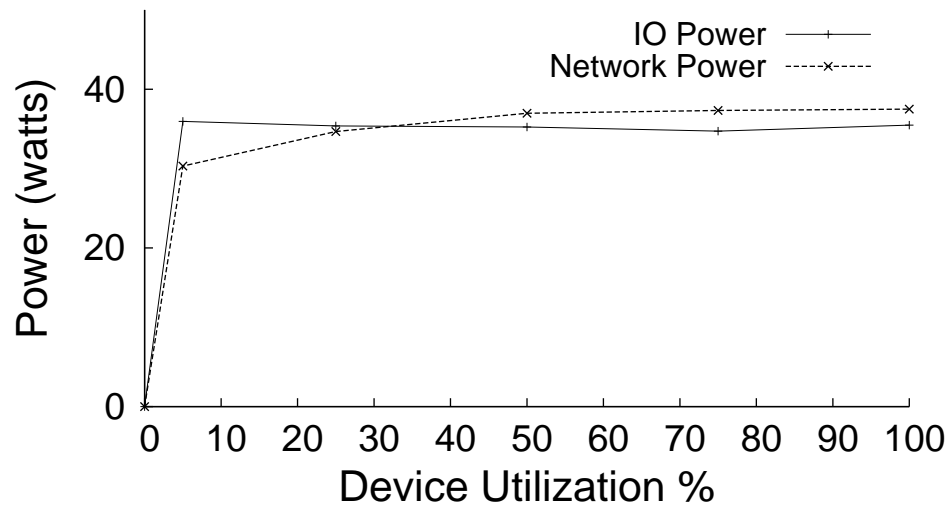


Figure 4.9. Effect of Power on Network and Disk IO throttle

The figure shows that while the network card used by the server shows a marginal energy proportionality of approximately 20%, the disk shows no energy proportionality whatsoever. It consumes 37W power when active and 0 when inactive. As the Dell PowerEdge R720 is amongst the high end modern server found at the time of writing, this experiment shows the deficiency of an energy proportional hardware design for many computing components. Interestingly the new disk technology of Solid State Drives (SSD) are not energy proportional either. The power consumption here is shown for the individual devices and not the server.

CHAPTER 5

EVALUATION

This chapter evaluates the effect of various design metrics on the performance of our application with the different policy variants. The various experiments performed below test the validity of our assumptions regarding how each design space constraint affects application runtime (performance). Each section explains what experiment was performed followed by results in the form of graphs comparing experiment completion time and energy consumed.

5.1 Sequential Write With Rapid S3 Transitions

Before diving into my experiments regarding energy agile designs, it is important to validate our assumptions as mentioned before regarding the usefulness of the ACPI S3 states compared to the active power capping mechanisms. Since my energy policies use the ACPI S3 state in all the policies to manage server power states, a worker node may become active and inactive several times during the span of an application while dealing with large amounts of data. It is important therefore to explore the effects of such rapid transitioning on data intensive applications.

Therefore to show the effects of S3 transitions on data intensive applications, an experiment was designed to disrupt disk writes by periodic suspensions into S3. A simple application was designed to sequentially write 32GB of random data to a single file on a SuperMicro server with 1TB disk and a dual core atom processor. While the application wrote data, the node was periodically put to the inactive S3 state and the total time of writing the file noted down. The same experiment was conducted first

with no suspension to S3, and then with decreasingly smaller periods (more rapid transitions) and the time for the experiment and each S3 transition noted down.

Figure 5.1 shows the total write time of for the file and the write time excluding the time taken to put the servers into the S3 transitions. As the figure shows, the time taken to write the 32GB file (y-axis) excluding the time taken to transition into and out of the state remains largely the same despite more S3 transitions due to a reducing period (x-axis). As the y-axis is in minutes, it does not show the negligible difference in the write times of the experiments. Nevertheless, this shows that S3 is an effective power state to manage servers as it does not interfere with data intensive applications.

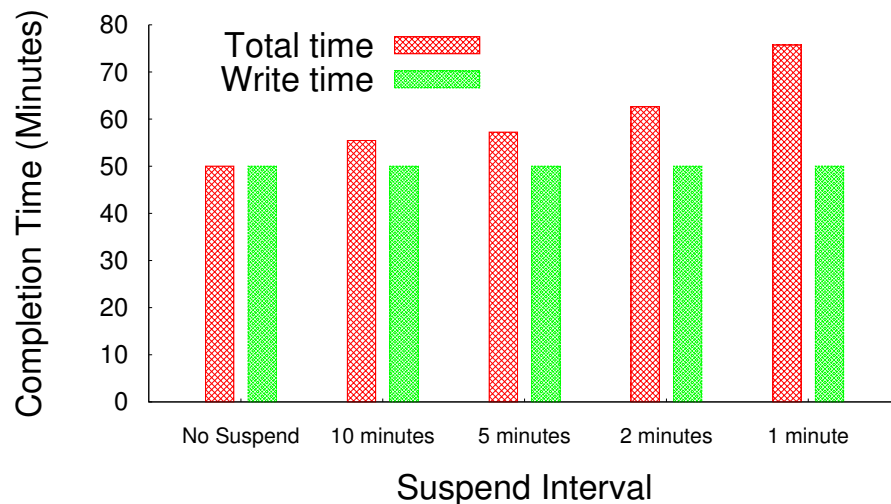


Figure 5.1. Effect of S3 transitions on Write a Sequential File

5.2 Energy Agile Performance

As the aim of my work is to design policies for energy agile applications, my work should be validated for energy agility. Reflecting back to the introduction of energy agility, it is the amount of computations done per energy available. Therefore the energy agile policies should be able to deal with a varying amount of available energy,

seeking to maximize application performance each time. This section shows such an experiment with a stable but varying energy budget available for each policy variant.

This section shows the performance of the various energy agile policies developed in my work. This experiment is also important as it shows applications can be enabled to be energy agile and work with varying or below peak requirement power signal by implementing hardware management policies and tweaking the application itself. The experiment shows different runs of the All to All Internode Communication application described in chapter 3. The experiment is run with a total of 50GB data distributed amongst 5 isolated Dell PowerEdge R720 worker nodes in the MGHPCC cluster (figure 4.4). This being a data intensive application, the network speed or CPU is never the bottleneck. The application speed is instead dictated by the disk speed.

Figure 5.3 shows the completion time of the application on the y-axis and the power budget available to the application on the x-axis. The performance of the application can be judged by it's completion time i.e the lower the better. All policies therefore exhibit a decreasing power performance as the application power budget decreases. Power budget as shown on the x-axis shows the average power delivered to the application platform throughout the course of the experiment. In this case, this was done by a flat input power curve at the specified power budget level as shown on the axis. Each policy behaves uniquely according to the graph and for the given conditions of transition time 10 seconds and not a lot of power fluctuations, blink outperforms the policies for the most part. The behavior of each policy is explained below.

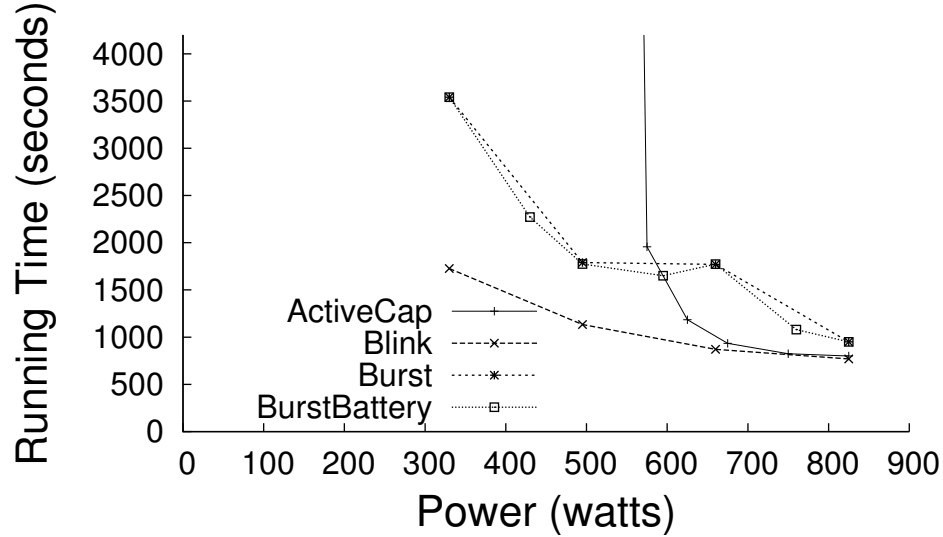


Figure 5.2. Power Budget Performance

Active Power Cap. Active power capping simply maintains the power budget by dividing the available power budget equally amongst all the nodes and power capping those nodes. As explained in previous sections, the cpu performance degrades massively with active power capping as it is the largest contributor to the servers dynamic power range. Hence as the cpu performance degrades quickly with a reducing power cap, it becomes the bottle neck and the application performance degrades severely. As shown in the graph, active power capping performs perhaps the best, alongside blink when the power budget is high, but degrades very quickly as it decreases, becoming the worst around the power budget 600W as its completion time approaches infinity.

Blink. Blink performs the best for these experiment conditions. The reason being that since blink keeps all the servers activated together, it does not need to save outbound buffers to disk incase a server is inactive. Hence less random access to the disk allows for a faster execution time. Same is true for active power capping as well. Blink here is run for an interval of 2 minutes with the duty cycle (chapter 3, subsection blink) being dictated by the power budget. Interesting to note here that

below the power budget of 330W, no other policy except blink allows the application to continue work as less than two active servers can be kept active which does not allow communication. Blink performs the best because the application transition time is assumed to be 10 seconds, which although realistic for some servers (Low performance super micro nodes), is not true for all.

Burst. The burst policy shows middle ground performance for these experiment conditions. As the power budget decreases such that fewer worker nodes can be concurrently kept active, burst performance decreases as the nodes have to go through more transitions in order to complete their NC2 links. The noticeably low difference in performance for power budget 445W (three active nodes) and 560W (four active nodes) is because there is not a big difference in the number of transition steps the nodes go through for both these power budgets.

Burst Battery. Burst battery is the same as burst, but with a battery to capitalize the wasted (overhead) power and use it productively for burst. Hence its performance is the same as that of burst, but with improvements where burst wasted power i.e in between power budget points where burst can use the whole budget to support a certain number of nodes (2,3,4 or 5). Hence the performance of burst battery is better than burst between these points as shown by the graph.

Similar to the graph above, the graph below shows the performance of each policy variant in terms of energy consumed to complete the available work. Interestingly it shows that the energy consumed varies a lot for different policy variant. Particularly for active power capping, energy consumed approaches infinity after a certain energy budget threshold (595W). Blink sticks to an average energy consumption level with not a lot of variation. Burst and Battery Burst largely follow the same pattern with a higher variation. The reason behind this is that although the nodes are using a smaller energy budget, they take more time to complete resulting in a larger completion time while using around the same energy. This does not hold for active power capping

because too much energy is wasted as base energy for some components which are not energy proportional as discussed in Chapter 3.

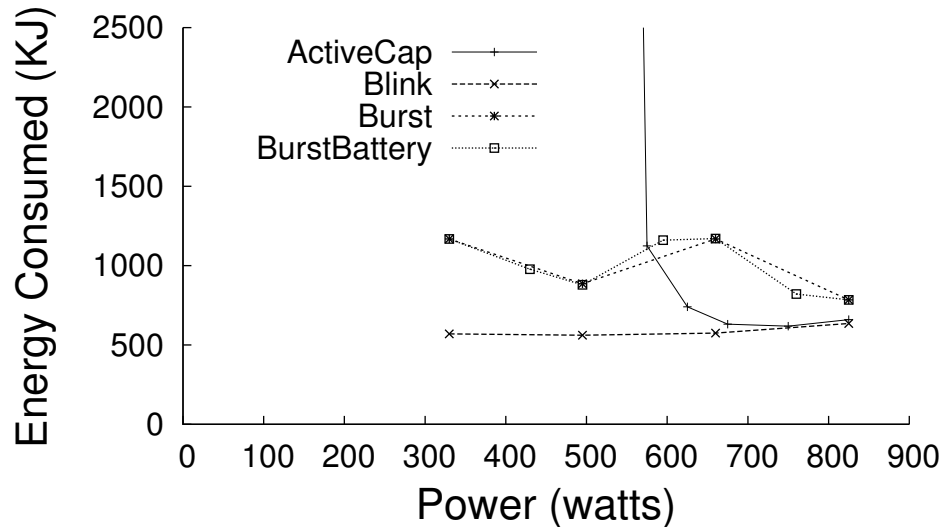


Figure 5.3. Power Budget Performance

5.3 Effect Of Transition Time

The main difference between my energy policies is the number of times the nodes are made to go through the ACPI S3 states. Since burst seeks to minimize the total number of transitions, it performs well if the transition time cost is very high. For blink however, the higher the transition time, the more blink cycles it takes to complete the work since each cycles does lesser work.

Typically the transition time is not symmetric i.e servers take more time going into S3 than coming out of it, but for the sake of simplicity I assume symmetry for my work. Burst is designed to keep the servers with the highest priority running, while transitioning only the lower priority servers to S3 when power changes. It results in fewer transitions than blink where every node is transitioned to S3, every duty cycle. Hence burst pays a lower penalty of transition time than blink. While transitioning between states, the servers consume power equivalent to when they were active, while

doing no productive work. Hence if the transition time is large, the servers could incur a significant increase in completion time, especially for blink.

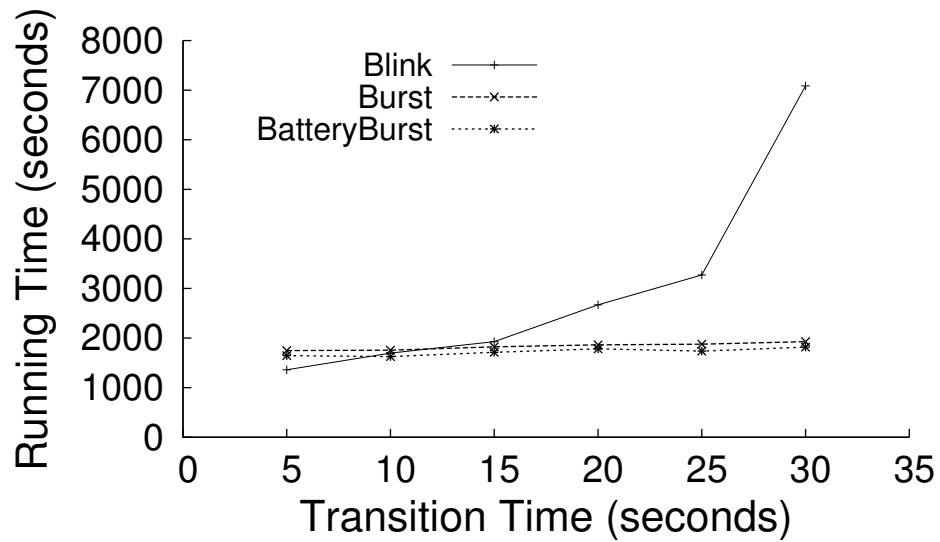


Figure 5.4. Transition Time Analysis

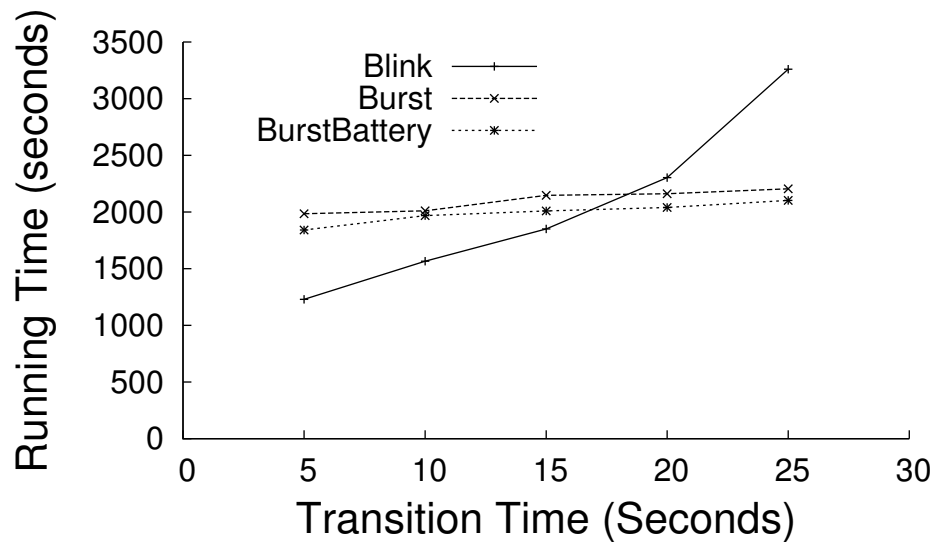


Figure 5.5. Transition Time Wind Analysis

The graph 5.4 shows the performance of the three policy variants as transition time is varied. The X-axis shows the transition time in seconds, the Y-Axis shows the completion time for the experiments in minutes. The figure 5.4 shows stable power

while figure 5.5 shows wind power. The average power delivered to the application in both cases is 5 minutes and the blink interval is kept at 75 seconds. As the transition time increases, the performance of burst more or less remains the same, with a very small increase. Since we assume a little wasted energy, batteryburst performs a little better than burst. The performance of blink on the other hand degrades very fast as the transition time increases because the productive work done in each blink interval reduces. Variations in the wind power impact application performance more for burst than for blink as the cross over transition time increases from 10 seconds to 18 seconds.

The figures below show the same experiments with energy consumed per experiment on the y-axis rather than the runtime per experiment. Both pairs of graphs show the same relationships with different metrics.

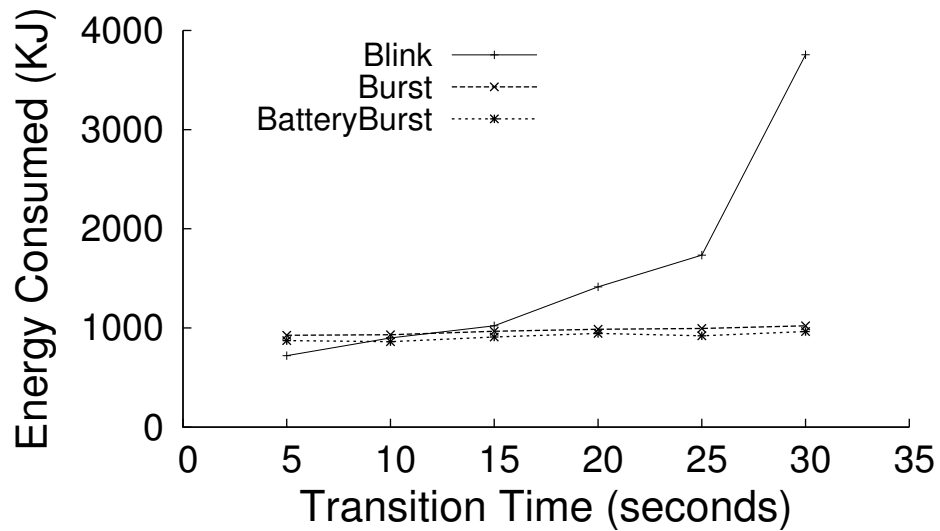


Figure 5.6. Transition Energy Analysis

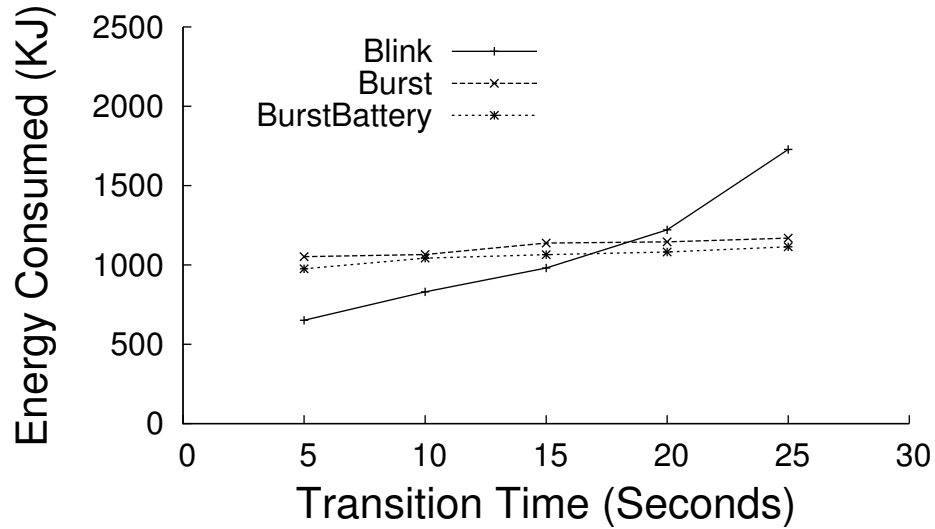


Figure 5.7. Transition Energy Wind Analysis

5.4 Effect Of Energy Buffer Size

Energy budget can determine the energy agility of a system, however things change a little when you have an energy storage available where you can store part of the energy budget for later use. Of the policies we have developed, only blink and battery burst use an energy buffer. Since the amount of energy storage can impact how each of these policies perform relative to each other, we examine this effect in this section. To analyze this effect, experiments were again conducted with 50GB data spread across 5 nodes with a power budget of 600W and transition time 10 seconds. The energy storage capacity was then varied and the completion time recorded.

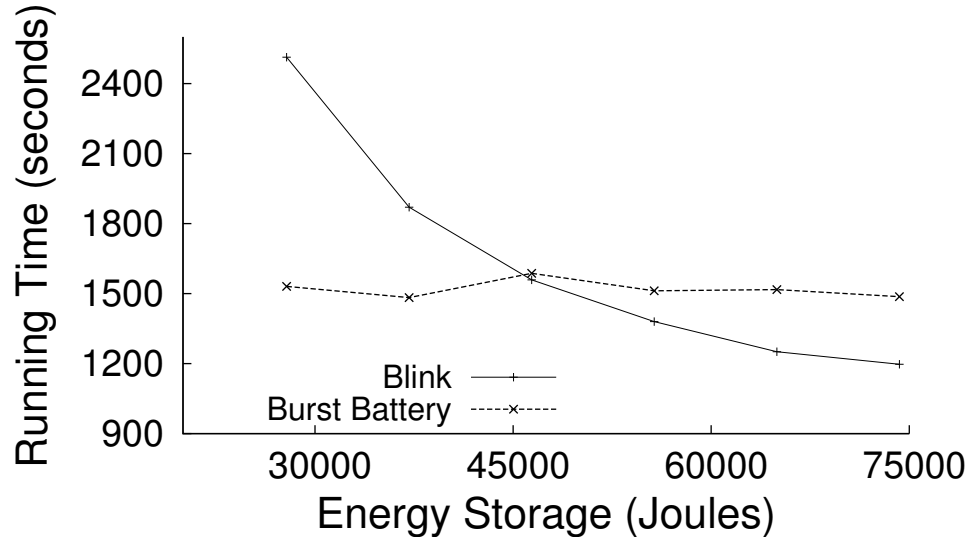


Figure 5.8. Energy Buffer Size Variation

Figure 5.8 the running time for the application on the y-axis against different energy storage capacities on the x-axis. It shows that for a very small buffer capacity, battery burst outperforms blink. However there is an exponential relationship between blink and buffer capacity with blink running time approaching infinity as the buffer approaches zero. Hence as the buffer capacity increases, blink's performance improves and it performs better than battery burst. Battery burst's performance remains more or less the same as the energy buffer capacity varies. This can be explained by the fact that battery burst does not use its stored energy until the buffer is completely full. For a larger buffer capacity, filling up the buffer with wasted power, which may be negligible takes a longer time, and may not fill up at all. Hence for the run time of the application, a larger buffer may be used fewer time because of the longer charge time, resulting in less benefits. A way around this might be by employing a similar duty cycle as in blink for battery burst as well.

5.5 Impact of power variation

Energy agility requires maximizing performance despite power signal variations. Although this does apply in the context of insufficient but stable power for the data centers, energy agility is more meaningful for practical varying power signals such as wind power. Experiments up till now have looked at stable power signals below the peak average of the application. Although this does show agility and the ability of the application to work with insufficient power, it does not reflect widely varying input power to show energy agility. Hence to analyze the effect of a realistic power signal on the experiments a wind power signal (Figure 3.8), generated from real wind data from around Amherst, MA was used as input power for the application. The application was then run for different transition times using the policies developed. Since active power capping has limited usage when the power drops significantly below the application average, it is not included in this experiment.

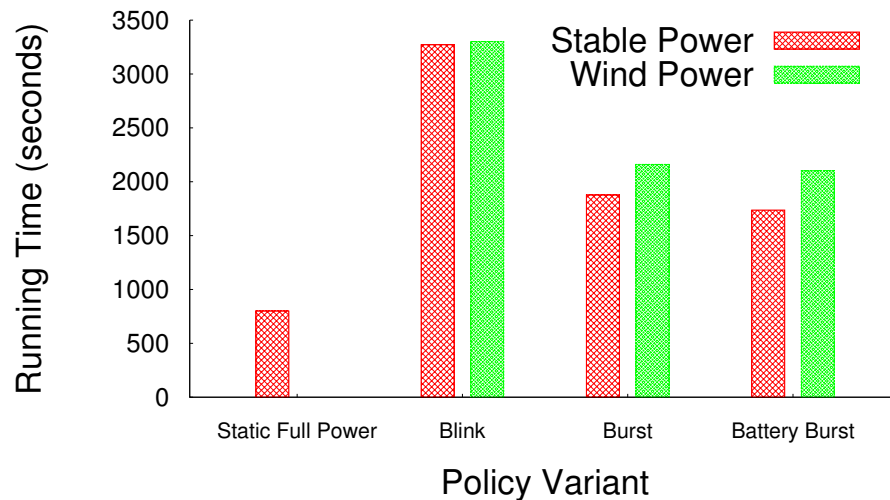


Figure 5.9. Impact of varying power on policy performance]

Figure ?? shows the relative performance of each policy variant on stable and wind power, showing how well each policy adapts to variations in the power signal. The x-axis shows the policy variants and the y-axis shows the completion time for

each policy on stable and wind power which both deliver an average power of 530W to the application. The blink interval was kept at 75 seconds and the transition time 25 seconds. This graph shows two things. First the performance degradation of blink vs burst and battery burst is very little due to power signal transitions. Second, for realistic transition times of 25 seconds or greater in today's servers, blink does not perform very well. Burst and battery burst have a much lower completion time in comparison. The graph below once again shows the same relations

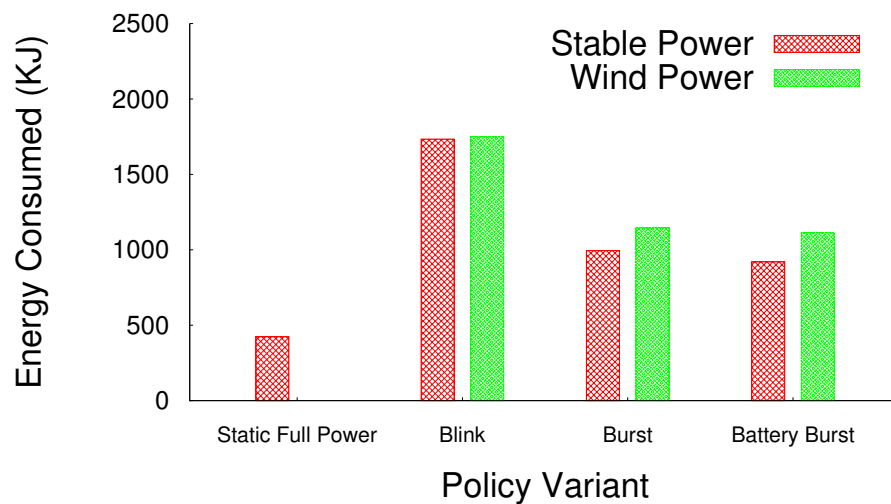


Figure 5.10. Impact of varying power on policy energy

Once again, we show the same experiment in terms of energy consumed to complete the application with each policy variant under the same conditions explained above. The energy graph is shown below.

5.6 Effect of Job Size

Transition time plays an important role in determining the performance of the policy variants. The previous section saw the direct effect of transition time on the application performance. Applications can also indirectly be impacted by the effect of transition time through job size. A larger job size would mean that blink has to go

through more cycles to complete the same work leading to lower performance. Hence one of the important metric which we found to significantly affect relative policy performance is the size of the job at hand. To show the effect, we carried out the experiment as shown below.

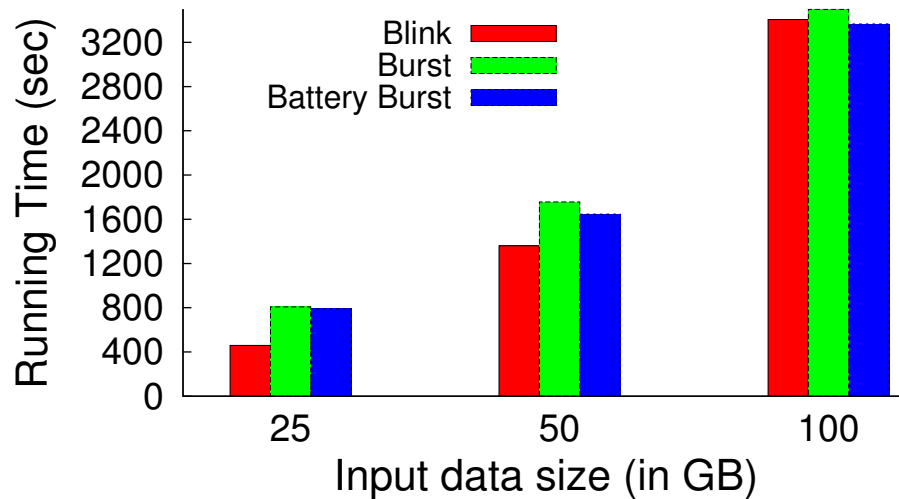


Figure 5.11. Impact of job size on policy performance

The figure 5.11 shows the performance of each policy variant for different job size. The x axis shows the size of the data distributed amongst the nodes and the y axis shows the completion time in seconds. Each colored bar represents a different policy variant. The graph shows that as the job size is increased, blink has to go through more cycles to complete the job, leading to a greater transition time overhead and decreasing its relative performance when compared to burst. Battery burst on the other hand makes use of more wasted energy and slightly improves performance.

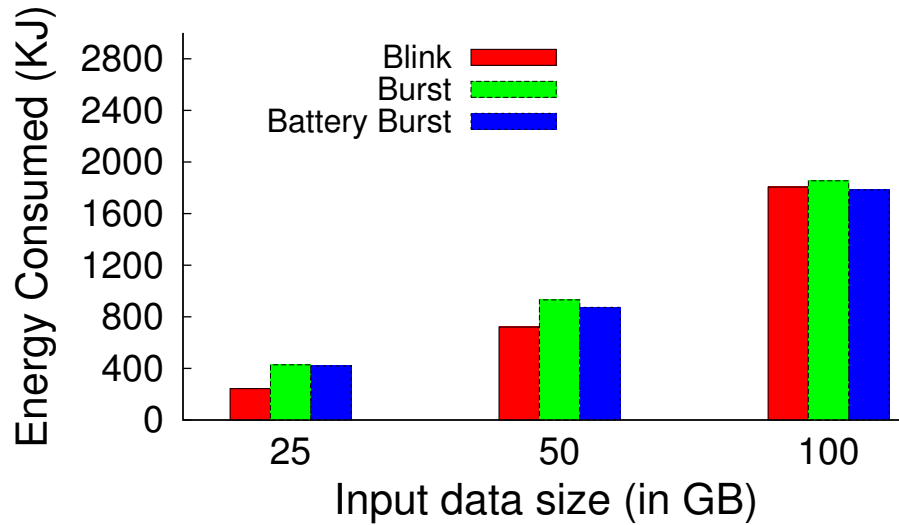


Figure 5.12. Impact of job size on policy energy

5.7 Experiment Results

The experiments conducted in this chapter show the impact of different design space metrics on the application performance in terms of energy consumption and time taken to complete. Each experiment illustrates a distinct point on how energy agile design can be affected by certain conditions. In this section we make generalizations on how each design metric may impact a general energy agile design. In all the experiments we see that battery burst will perform slightly better than burst if wasted energy is available. As such the generalizations for burst can be extended to battery burst.

Figure 5.3 shows that active power capping is only useful up to a certain threshold of a stable power signal. Below that threshold, too much energy is used as base power to power the other components of the server which may be being used by the application but are not power proportional such as disk, network, etc ???. As a result, too much base power is used for such devices and the CPU being the major contributor in active power cap management takes too much of a performance hit.

Hence the application becomes too slow, approaching a completion time of infinity as the energy budget approaches a certain threshold, 595W in this case. This threshold depends on the number of nodes and the power they consume when not doing any work.

Secondly, our comparison of the performance of the energy policies blink and burst for different transition times showed that for shorter transition times blink typically performs better than burst. However for larger transition times, burst can perform up to 4X faster than blink. The point where blink becomes better than burst typically depends on the variability of the power signal, the platform being used and the energy budget being delivered by the power signal. The comparison of transition times of the policies of blink and burst for wind and stable power signals further shows that blink relatively performs better than burst, using 18% less energy to complete in our particular case.

Third, the size of the job can play an important role in the relative performance of energy agile policies. As shown by figure 5.11, when shuffling 100GB of data blink is 4% faster than burst while when shuffling a smaller 25GB dataset blink is 45% faster than burst.

We see that for practical transition times of around a few minutes as seen in real high end servers deployed in data centers, the energy policy burst will typically perform better than blink. As explained above, the performance of blink decreases dramatically as the transition time increases, leading to a smaller portion of the blink interval being spent in doing actual work.

CHAPTER 6

CONCLUSION

The application I have worked with captures the bottleneck of typical data centric applications. As such the lessons learnt from my experience in designing and developing a solution to the All to All Internode Communication application are applicable to a broader set of data intensive batch applications, such as Sort or MapReduce. The lessons are summarized below:

Inactive power capping is useful, despite it's overhead. Previous work has largely focussed on active power capping [8, 22, 23, 26], since inactive power capping is not suitable for all workloads. One example would be online data-intensive (OLDI) workloads, which may require immediate access to some data which may lie in any of the server at any given time and cannot afford the overhead involved with inactive power capping [22, 23]. On the other hand, my work shows that for data intensive parallel batch applications, which may include internode communication as a subtask, inactive power capping is much more efficient than using active power capping as it uses more power to do useful work, offsetting the transition overheads which come with inactive power capping mechanisms. Additionally, where applications can use active power caps to satisfy caps as low as 50% peak power, they slow down the server progress such that it is not beneficial due to high server idle powers.

Blinking performs better when Coordination is necessary. As my experiments show, blinking does incur a high transition time penalties at short intervals, however it does not affect the application's pattern of I/O as all the servers are always concurrently active and inactive. Other policies which deactivate subsets of the

servers potentially alter the application's I/O patterns degrading their performance by changing sequential I/O to random I/O as all the servers are not concurrently active. For example MapReduce, Sorting and other "big data" tasks have frequent periods of large scale coordinated data movements. Burst however performs better where the parallel subtasks of an application do not require coordination, reducing the transition overheads.

As job size increases, relative performance of blink decreases. The experiments in this chapter show that the performance of blinking is dependent on job size among other factors. As job sizes increases, more blink cycles are required to complete the same amount of work, leading to a higher amount of time spent to complete work. For burst on the other hand the same number of transitions are required to complete a longer job size, leading to a lower proportion of time spent in transitioning the nodes, increasing its relative performance.

For more practical transition times, burst performs better. As shown in the experiments a larger transition time decreases the amount of productive work done in each coordinated blink cycle, leading to relatively lower blink performance.

BIBLIOGRAPHY

- [1] Apple and the environment. <http://www.apple.com/environment/renewable-energy/>.
- [2] Report to congress on server and data center energy efficiency: Public law 109-431. Tech. rep., U.S. Environmental Protection Agency, August 2nd 2007.
- [3] Agarwal, Y., Hodges, S., Chandra, R., Scott, J., Bahl, P., and Bahl, P. Somniloquy: Augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX Symposium on Networked System Design and Implementation* (Boston, Massachusetts, April 2009), pp. 365–380.
- [4] Andersen, D., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. FAWN: A Fast Array of Wimpy Nodes. In *SOSP* (October 2009).
- [5] Chang, Fay, Dean, Jeffrey, Ghemawat, Sanjay, Hsieh, Wilson C., Wallach, Deborah A., Burrows, Mike, Chandra, Tushar, Fikes, Andrew, and Gruber, Robert E. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 15–15.
- [6] Cochran, R., Hankendi, C., Coskun, A., and Reda, S. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *MICRO* (December 2011).
- [7] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI* (December 2004).
- [8] Fan, X., Weber, W., and Barroso, L. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)* (San Diego, California, June 2007), pp. 13–23.
- [9] Faruqui, A., Hledik, R., and Tsoukalis, J. The power of dynamic pricing. In *Electricity Journal*, 22(3) (February 2009), pp. 42–56.
- [10] Finley, K. Facebook Says its New Data Center Will Run Entirely on Wind. In *Wired* (November 13th 2013).
- [11] Gandhi, A., Harchol-Balter, M., Das, R., Kephart, J., and Lefurgy, C. Power Capping via Forced Idleness. In *Weed* (June 2009).

- [12] Goiri, I., Le, K., Haque, M., Beauchea, R., Nguyen, T., Guitart, J., Torres, J., and Bianchini, R. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *SC* (April 2011).
- [13] Goiri, I., Le, K., Haque, M., Beauchea, R., Nguyen, T., Guitart, J., Torres, J., and Bianchini, R. Greenslot: Scheduling energy consumption in green datacenters. In *Proceedings of the International Conference for High Performance Computing* (Seattle, Washington, April 2011), Networking, Storage and Analysis (SC).
- [14] Goiri, I., Le, K., Nguyen, T., Guitart, J., Torres, J., and Bianchini, R. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *EuroSys* (April 2012).
- [15] Hamilton, J. I Love Solar Power But... <http://perspectives.mvdirona.com/>, March 17th, 2012.
- [16] Irwin, D., Sharma, N., and Shenoy, P. Towards continuous policy-driven demand response in data centers. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Green Networking (GreenNets)* (Toronto, Canada, August 2011), pp. 19–24.
- [17] J.Koomey. Data center electricity use 2005 to 2010. Tech. rep., Analytics Press,, August 1st 2011.
- [18] Koomey, Jonathan G., Belady, Christian, Patterson, Michael, Santos, Anthony, and Lange, Klaus-Dieter. Assessing trends over time in performance, costs, and energy use for servers. Tech. rep., Lawrence Berkeley National Laboratory and Stanford University, August 17, 2009.
- [19] Lakshman, Avinash, and Malik, Prashant. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44, 2 (Apr. 2010), 35–40.
- [20] Lean, Hooi Hooi, and Smyth, Russel. Are fluctuations in us production of renewable energy permanent or transitory? In *Applied Energy* (January 2013), vol. 101, pp. 483–488.
- [21] Lefurgy, C., Wang, X., and Ware, M. Server-level Power Control. In *ICAC* (February 2007).
- [22] Lo, D., Cheng, L., Govindaraju, R., Barroso, L., and Kozyrakis, C. Towards energy proportionality for large-scale latency-critical workloads. In *International Symposium of Computer Architecture (ISCA)*, p. June 2014.
- [23] Meisner, D., Sadler, C., Barroso, L., Weber, W.-D., and Wenisch, T. Power management of online data intensive services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)* (San Jose, California, June 2011), pp. 319–330.

- [24] Miller, R. Data Centers Scale Up Their Solar Power. In *Data Center Knowledge* (May 14th 2012).
- [25] Miller., R. Data centers scale up their solar power. In *Data Center Knowledge* (May 14th 2012).
- [26] Ranganathan, P., P. Leech, D. Irwin, and Chase, J. Ensemble-level power management for dense blade servers. In *Proceedings of the Thirty-third Annual International Symposium on Computer Architecture (ISCA)* (Boston, Massachusetts, June 2006), pp. 66–77.
- [27] Rasmussen, A., Conley, M., Kapoor, R., Lam, V.T., Porter, G., and Vahdat, A. Themis: An I/O-Efficient MapReduce. In *SoCC* (October 2012).
- [28] Rivoire, S., Shah, M., and Ranganathan, P. JouleSort: A Balanced Energy-Efficient Benchmark. In *SIGMOD* (June 2007).
- [29] Sharma, N., Barker, S., Irwin, D., and Shenoy, P. Blink: Managing Server Clusters on Intermittent Power. In *ASPLOS* (March 2011).
- [30] Stewart, C., and Shen, K. . some joules are more precious than others: Managing renewable energy in datacenter. In *HotPower* (October 2009).
- [31] Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., and Zhu, X. Delivering energy proportionality with non-energy-proportional systems: Optimizing the ensemble. In *Proceedings of the Workshop on Power-aware Computing and Systems (HotPower)* (San Diego, California, December 2008).
- [32] Tsirogiannis, Dimitris, Harizopoulos, Stavros, and Shah, Mehul A. Analyzing the Energy Efficiency of a Database Server. In *SIGMOD* (June 2010).
- [33] von Meier, A. *Electrical Power Systems: A Conceptual Introduction*. John Wiley and Sons, Inc., 2006.
- [34] Wierman, A., Z. Liu, I. Liu, and Mohsenian-Rad, H. Opportunities and challenges for data center demand response. In *IGCC* (June 2014).
- [35] Z.Liu, Wierman, A., Chen, Y., Razon, B., and Chen, N. Data center demand response: Avoiding the coincident peak via workload shifting and local generation.