

DESIGN AND EVALUATION OF A POWER-AWARE PARALLEL I/O SYSTEM

BY

KARTHIK PATTABIRAMAN

B.Tech., University of Madras, 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

Power consumption is a major concern for high-performance computing systems. Of the components in such a system, the storage subsystem consumes a significant portion of the total power. In this thesis, we will examine techniques for reducing the power consumption of high performance parallel I/O systems. The underlying assumption is that the system is composed of disks that can be spun up and spun down on demand. Parallel I/O workloads typically vary over time and this can be utilized to shut down disks during periods of low activity and spin them up again when the load increases. This must be done in a scalable, consistent fashion to guarantee that response times do not exceed a certain bound. This thesis will examine the design space of configuration options of the system (stripe depth, number of replicas etc.) and also the dynamic adaptation mechanism (number of disks to keep active for a given load). It will evaluate the system under bursty, synthetic workloads and real workloads using analytic modeling and simulation.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Daniel A. Reed, for his support and guidance throughout the course of this project. He has been extremely patient and helpful and never failed to guide me when I was in the need for advise. I would like to thank Dr. Celso Mendes for the many insightful discussions regarding this project. I would also like to thank Ying Zhang for helping me with various technical issues involved in this project. I would also like to thank Chantelle Houglund, for providing me technical and administrative support and Wei Deng for providing cluster support to run my simulations.

The I/O traces used in this thesis were collected by Michael Treaster for his M.S thesis work. I would like to thank him for making them available to me and his ready cooperation in working with me to resolve technical issues related to them. I would also like to thank Charng-da Lu, a fellow student in the Pablo group for being the sounding board for many of my ideas, and other members of the Pablo group for making my work fun and enjoyable.

On the personal side, I would like to thank my wife Padmapriya Kandhadai, for being a source of strength and inspiration to finish this thesis. I would also like to thank my parents and friends for their understanding and encouragement.

This work was partly supported by the Department of Energy(DOE) under contract numbers W-7405-ENG-36 and DE-FC02-01ER25488 and by the National Science Foundation (NSF) under grant EIA-99-75248.

Table of Contents

List of Tables.....	vii
List of Figures.....	viii
Chapter 1 Motivation	1
1.1 Power-Aware High-Performance Computing	1
1.2 Power-Aware Parallel File Systems	2
Chapter 2 Related Work	5
2.1 Parallel File Systems	5
2.2 Disk Power Management	6
2.2.1 Mobile-devices	6
2.2.2 High-Performance Systems	7
2.3 Power-Aware High-Performance Systems	10
Chapter 3 System Model.....	12
3.1 Overview	12
3.2 System Description	12
3.3 Qualitative Analysis	14
3.4 Quantitative Analysis	15
3.5 Discussion	18
3.5.1 Request Rate Versus Number of Replicas	19
3.5.2 Request Length Versus Number of Replicas	26
Chapter 4 Implementation	29
4.1 Overview	29
4.2 Architecture	29
4.2.1 Request-Generation Sub-System	30
4.2.2 Topology and Request-Routing Sub-System	32
4.2.3 Adaptation Sub-System	35
4.2.4 Statistics Gathering and Link Sub-Systems	36
4.3 Algorithms	37
4.3.1 Request-Generation Algorithms	37
4.3.2 Request-Routing Algorithms	40
4.3.3 Adaptation Algorithms	41
4.4 Summary	43

Chapter 5	Experimental Infrastructure and Results	46
5.1	Overview	46
5.2	Experimental Infrastructure	46
5.2.1	Simulation	46
5.2.2	Run Length Control and Transients	47
5.2.3	System Parameters	48
5.3	Applications	50
5.3.1	ESCAT	51
5.3.2	SPPM	51
5.4	Results	52
5.4.1	b-Model Results	52
5.4.2	Application Results	59
5.4.3	Summary	64
Chapter 6	Conclusions and Extensions	70
6.1	Overview	70
6.2	Conclusions	70
6.3	Extensions	71
References		74

List of Tables

3.1	Parameters in the Model	15
3.2	Values used for Analysis	18
5.1	Parameter values used in simulation	49

List of Figures

3.1	Request Rate Vs Number of Replicas for a request size of 8 blocks . .	20
3.2	Request Rate Vs Number of Replicas for a request size of 64 blocks .	21
3.3	Request Rate Vs Number of Replicas for a request size of 256 blocks .	22
3.4	Request Rate Vs Number of Replicas for a request size of 1024 blocks	23
3.5	Request Length Vs number of replicas an arrival rate of 20 requests/sec	24
3.6	Request Length Vs Number of Replicas for an arrival rate of 40 re- quests/sec	25
3.7	Request Length Vs Number of Replicas for an arrival rate of 60 re- quests/sec	26
3.8	Request Length Vs Number of Replicas for an arrival rate of 80 re- quests/sec	27
4.1	The various sub-systems and their interactions	30
4.2	Block Diagram of the Topology and Request-Routing Sub-system . .	33
4.3	Traces generated by the b-model $n = 1000$, $N = 10000$ for different values of b	44
4.4	Monitor parameters and their relationship	45
4.5	Modes of Operation of the System with respect to the MinMax Controller	45
5.1	b-model Results for Rate = 10 Requests/sec, Length = 1024 blocks .	53
5.2	b-model Results for Rate = 20 Requests/sec, Length = 1024 blocks .	54
5.3	b-model Results for Rate = 30 Requests/sec, Length = 1024 blocks .	55
5.4	b-model Results for Rate = 40 Requests/sec, Length = 1024 blocks .	56
5.5	b-model Results for Rate = 50 Requests/sec, Length = 1024 blocks .	57
5.6	b-model Results for Optimal Utilization	58
5.7	Distribution of read requests over time in ESCAT	60
5.8	Distribution of read requests over time in ESCAT for the first 10 seconds	61
5.9	Figure showing the adapation of the system to the escat trace for different group sizes	65
5.10	ESCAT Results for Response-Time	66
5.11	ESCAT Results for the Response-Time Power Product	66
5.12	Distribution of read requests over time in sPPM	67
5.13	Breakdown of read requests in sPPM for each processor	67
5.14	Figure showing the adapation of the system to the sppm trace for different group sizes. The number of nodes involved in I/O is fixed at 8	68
5.15	SPPM Results for Response-Time	69
5.16	SPPM Results for the Response-Time Power Product	69

Chapter 1

Motivation

1.1 Power-Aware High-Performance Computing

Traditionally, specialized parallel computer architectures, ranging from vector supercomputers to massively parallel multiprocessors, have dominated the supercomputing domain. Over the last decade, a new platform for parallel computing has emerged. Networks of Workstations or NOWs as they are popularly called, have emerged as a viable platform for certain classes of applications requiring high performance computers, at substantially lower costs [1]. NOWs employ a large number of smaller scale multiprocessors connected over a high performance network.

The Beowulf project [2] takes the concept of NOWs to an extreme by making use of mostly commodity off-the-shelf components (COTS). Beowulfs also use commodity software, enabling them to leverage the large amount of development effort in the scientific community. Beowulfs run standard operating systems like Linux/Windows, use the Message Passing Interface (MPI)[3] programming model or the Parallel Virtual Machine (PVM)[4] programming model, and run parallel file systems like PVFS [5]. All these factors make the Beowulf the ideal platform from a price-performance point of view [6]. Today, it is possible for even small organizations to set up a Beowulf cluster for a fraction of the price of a large supercomputer.

The main problem with COTS components compared to specialized hardware components is their high rate of failure. As modern hardware components increase in complexity and capability, they draw more and more power and this in turn increases their temperature, making them more prone to failures. According to the Arrhenius's equation (applied to micro-electronics), the failure rate of a given system doubles with every 10 degrees Celsius rise in temperature. This has also been verified empirically using unpublished data from vendors by Feng et. al. [7]. Thus, to make high performance computers easier to maintain and administer, we must make them more reliable. This can be done by reducing the power consumption of these systems, which in turn, would lead to reduced temperature and reduced failure rate. This leads us to a new paradigm of high-performance computing called "low-power high-performance" computing.

1.2 Power-Aware Parallel File Systems

It has long been recognized that I/O is one of the main bottlenecks to the scalability of parallel scientific codes [8]. The reason for this is that processor and communication speeds have continued to increase, but I/O speeds have not kept up correspondingly. Furthermore, parallel applications which access gigabytes or even terabytes of data are becoming more and more common [9].

To improve performance, advances of I/O hardware and file system parallelism are of principal importance [10]. In the last few years, a wide variety of parallel I/O systems have been built and proposed [5; 11–17]. All these systems exploit parallel I/O devices i.e., partitioning data across multiple disks for parallelism in an attempt to deliver high I/O performance. However, a new class of multi-teraflop applications is emerging, with time-varying resource demands and changing access patterns. These applications manipulate multi-terabyte data sets, and their I/O demand cannot be

met by extant parallel file systems [18]. To support these emerging applications, next-generation file systems will have to stripe data over thousands of secondary and tertiary storage devices [19–21].

A high-performance system consisting of many thousands of disks will consume large amounts of energy, as a disk consists of moving components which consume power even when they are inactive. For example, the disk motor remains spinning even when there is no request to the disk. Laptops and other portable devices, which are energy-constrained, stop the motor and spin the disk down when it is idle for long periods of time, and spin it up again when a request is issued to it. Unlike laptop disks, server disks and disks used in high-performance workstations are not equipped with the capability to be spun-up and down on demand. The reason for this is that laptop disks are manufactured to withstand the stresses of spin-ups and spin-downs, but they have relatively low performance when server workloads are considered and their MTTF (mean Time To Failure) is an order of magnitude lower than server disks. Server disks, on the other hand, are designed for high performance and long MTTF, but they cannot withstand the mechanical stress of spin-ups and spin-downs.

However, there has been some recent work which make a case for using disks with dynamic spin-up/spin-down capabilities in web servers [22]. Since power-managed web servers can be packed together in dense configurations in data-centers, leading to better utilization of floor-space, lower energy bills and cooling requirements, they can directly impact the financial aspects for the data center [23]. As a result, the authors of [23] believe that there is great incentive for server disk manufacturers to make disks which can withstand the mechanical stress of spin-ups and spin-downs without impacting their MTTF (Mean Time To Failure) [23]. At any rate, power is as important a factor as performance and reliability in web servers, and one can be traded off for the other two.

We believe that like web server farms, high-performance systems should also use

disks which can be spun-up and spun-down on demand for low power. Like server systems, high-performance systems are also usually over-provisioned to meet the maximum demands of parallel scientific applications. The I/O requirements of these applications are highly bursty and irregular and the average case requirement is much lower than their worst-case demands [9]. The typical I/O pattern of these applications consists of short periods of high demand followed by long periods of little or no activity [24].

The variability of the I/O workload creates many opportunities for power management of disks in high-performance systems. However, this is different from battery-powered platforms which spin the disk down if it is idle for long periods of time. These systems mostly deal with interactions with a single user, who might be willing to tolerate some performance loss in return for longer battery life. Moreover, response-time and not throughput is usually the most important metric in these systems. A high-performance storage system, on the other hand, often serves many users or applications with varying requirements, and users may not be willing to tolerate performance losses of more than 5-10 %. Furthermore, these are complex systems consisting of multiple disks, storage buffers, network interfaces and processors, and it is essential to consider the interaction of the power management policies of all these devices in a global fashion.

In this thesis, we describe the design, implementation and evaluation of a power-aware parallel I/O system, which applies dynamic spin-up/spin-down techniques to multiple disks. It considers the trade-offs in disk striping and mirroring of data, and intelligently chooses the right configuration for a given workload. It also dynamically adapts the system to the workload, achieving the desired performance with the least energy consumption. We evaluate this system for real and synthetic workloads.

Chapter 2

Related Work

In this chapter, we survey related work in parallel file systems and disk power management techniques for both mobile and server environments. We then survey power-conservation techniques in high-performance systems, other than in the storage system.

2.1 Parallel File Systems

Parallel File Systems have been developed for many commercial platforms by their respective vendors. These include PFS for the Intel Paragon [14], PIOFS and GPFS for the IBM SP [12], HFS for the HP Exemplar [11] and SGS for the SGI Origin [25]. These file systems provide high-performance and functionality for I/O intensive applications, but are available only on specific platforms. PVFS [5] is an open-source, freely available parallel file systems developed for Beowulf clusters running Linux.

A number of research projects in the area of parallel I/O exist, and they have developed parallel file system prototypes for use in their research. PIOUS [16] employs data declustering and transaction-based concurrency control, to provide scalable bandwidth in a parallel environment. Galley [17] is designed with the access characteristics of parallel application workloads in mind. It gives the application control

over disk parallelism and data layout, so that application developers can tailor it to their application's needs. It also provides support for common access patterns in specialized libraries.

PPFS [26] (Portable Parallel File System) is a parallel I/O library invoked by user I/O calls that offers control over a variety of input/output configurations such as disk striping factor, prefetch and caching policies. It was developed to explore the design space for scalable I/O systems, and relies on the underlying system software for physical I/O. It has the ability to dynamically adapt its policies to match the access patterns of the application.

PPFS 2 is the next generation of the PPFS file system, and is designed for I/O over computational grids [27]. It allows rule-based, closed-loop adaptive and interactive control of input/output systems on both parallel and wide-area distributed systems to support complex, irregular applications with data-dependent and time-varying resource demands.

2.2 Disk Power Management

2.2.1 Mobile-devices

Power management has traditionally been a concern for mobile devices like laptops and PDAs, where the aim is to maximize battery life. Power management for laptop disks is a well explored problem, and many techniques have been developed for it. Using the taxonomy introduced by Lorch and Smith [28], we can classify these techniques into two broad groups. The first group comes under the category of transition strategies, which deal with when the disk should be spun-up or spun-down to reduce power consumption. The second group comes under the category of load-change strategies, which deal with how the workload can be modified in order to achieve better power conservation.

Transition strategies spin down the disk if it is idle for a certain amount of time greater than the timeout. Greenawalt [29] studies the effect of changing the timeout on reliability, performance and power-consumption using an analytical model based on Poisson arrivals. Li et. al. [30] find that most of the energy consumed by a disk can be saved if the timeout was set to a few seconds, rather than a few minutes, based on real filesystem traces. Douglis et. al. [31] compare a threshold-based policy to an optimal policy and compare different algorithms for disk-spindown. Douglis, Krishnan and Bershad [32] describe an adaptive algorithm that dynamically varies the threshold parameter depending on previous observations about the workload. Helmbold et. al. [33] propose an adaptive scheme based upon machine-learning techniques to dynamically choose the best algorithm for setting the time-out from a list of candidate algorithms.

Load change strategies increase idle-periods in I/O request sequences so that the disk can be spun down for longer periods of time. This can be done either by delaying or prefetching requests. Weissel et. al. [34] propose a scheme which allows the application to participate in power-management by specifying a timeout and an abort mechanism. This allows greater flexibility on the part of the operating system to satisfy I/O requests, but involves changing the I/O interface to the application. Another scheme involves increasing the utilization of the disk when it is spun-up and deferring requests when it is spun-down. It was proposed by Papathanasiou et. al. [35] and requires the operating system to cluster I/O operations in time by aggressively prefetching data for synchronous reads and deferring of non-critical I/O operations.

2.2.2 High-Performance Systems

Recently, there has been a lot of interest in power management for high-performance storage systems [36–40]. However, most of these studies have focussed on web-server

or data center environments with transaction processing workloads. The only exception is the paper by Colarelli and Grunwald [37] which uses file-level traces from a supercomputing center for evaluation. This paper proposes a storage architecture called Massive Array of Idle Disks (MAID) in which power-managed disks are used for archival storage in lieu of tape devices. Here, the emphasis is on power-saving rather than on performance, and power management is done on a per drive basis. A drive that is idle for a certain period of time is powered down, and powered up again when a request is sent to it. The MAID system also designates a small number of drives as cache-drives and uses them to cache recently-used data. Read requests that miss in the cache are sent to the data drives, spinning them up again if they were powered down. Writes that miss in the cache are written to the cache, and written back to the data drives only when the data drives are powered up again due to read misses.

Though our system is also targeted for supercomputing centers, it differs from the MAID system in a number of ways. First of all, we do not target archival storage, and performance is still our main concern. We provide an analytical model for the performance degradation and ensure that it is bounded within a threshold. More importantly, we consider power management of all the drives in the system together, and do not individually set the policy for each drive. This enables us to coordinate the activities of all disks and avoid performance bottlenecks. Furthermore, we do not cache frequently accessed data in a small set of drives, as most scientific workloads exhibit little or no locality. This fact is reflected in the evaluation of the supercomputing workload with MAID, and the authors mention that caching provided them with little or no benefit.

We believe that the biggest contribution of our work is that we explicitly quantify the effects of striping on power and performance and analyze the tradeoffs involved. We provide a detailed analytical model for assessing the benefits of striping and for

choosing the stripe size. The MAID paper served as one of the main motivating works for our study of the effects of striping on power management.

Another work which is closely related to ours is Gurumurthi et. al. [38], which analyzes the trade-off between energy and performance by varying the stripe size for disk arrays. However, it is primarily concerned with transaction processing workloads, in which the idle periods are very small, compared to the time taken to spin-up or spin-down. As a result, they do not do any spin-up or spin-down of disks, but instead look at the access costs due to disk-head positioning overheads. However, scientific workloads consists of long, idle or low-activity periods followed by high-volume I/O bursts, and hence spin-up/spin-down is a viable option. Further, our system uses replication and has at least one replica active all the time, to deal with the presence of frequent but low-volume I/O requests. As a result, we can spin-down most of the disks during these low-activity periods.

Carrera et. al. [36] study different approaches to conserving energy in high-performance network servers. One of the schemes proposed called "Combined" involves combining high-performance and laptop disks, such that both sets of disks mirror the same data. They exploit offered load variations to keep only one set of disks powered on at any time. When the offered load is high, they power-on only the high-performance disks and when it is low, they power-on only the low-power disks. They also keep the two copies coherent by performing updates made to one set of disks to the other set immediately after powering it up.

Our system also keeps multiple copies of the data and exploits offered load variations to spin-up or spin-down disks. However, our scheme differs from theirs in two ways: First of all, we assume all our disks can be spun-up or spun-down, and we can have more than one replica powered up and load balance between them. Because we assume a read-only data set, we do not worry about keeping the replicas coherent. Secondly, we can have more than two sets of disks and spin them up and down on-

demand depending on the offered load. In this way, our scheme is a generalization of the scheme proposed by Carrera et. al. [36], but with the addition of load-balancing between multiple replicas.

Another approach to spinning-up and spinning-down disks is to use variable speed disks (ie) disks in which the speed of rotation can be varied. Gurusurthi et. al. [39] have shown that it is possible to save upto 60 % of the disk energy using this technique. This is especially true for transaction processing workloads, where the idle-time periods are not long enough to save power by spinning the disk down completely, and running the disk at multiple speeds(depending on the workload) may be the only way to save power. However, the workloads we consider have long enough idle-times and spinning down disks during these periods can lead to considerable power savings.

Zhu et. al. [40] explore various caching strategies to minimize the energy consumption of high-performance storage centers. They also spin-down disks when they are inactive and aggressively cache data to avoid reads from going to powered-down disks. They also investigate various write-back policies to avoid spinning up powered-down disks unless absolutely necessary. They show that an energy-aware cache replacement algorithm can conserve more energy than an energy-oblivious, but performance-optimal algorithm. We do not consider caching and prefetching in our scheme, though it would be an area of possible future expansion for our system.

2.3 Power-Aware High-Performance Systems

Finally, we look at related work in the area of power conservation in high-performance systems, not just storage systems. Pinheiro et. al. [41] dynamically power-on and power-off nodes in a cluster web-server depending on the incoming load. Chase et. al. [42] propose an energy-conscious, dynamic resource provisioning architecture for

web servers in which servers bid for energy and other resources based on the notion of utility. The design of our MinMax controller is motivated by their flop-flip filter which is used to smooth noise in the input workload. A study by Elnozahy, Kistler and Rajamony [43] looks at the trade-offs between using the power-on, power-off strategy proposed by [41] and doing dynamic voltage scaling (DVS) for the CPU. Elnozahy et. al. [44] use a feedback-driven control policy similar to ours to adapt the cluster to changes in the workload.

Green Destiny is a Beowulf cluster designed for low power and high density. It was designed and developed by Los Alamos National Laboratory, in conjunction with RLX Technologies. Its design and architecture is described in Warren et. al. [45]. It consists of compute nodes made from COTS parts mounted on motherboard blades called RLX Server-Blades. Each motherboard blade contains a 633-Mhz Transmeta TM 5600 CPU, 256 MB memory, 10 GB hard disk and three 100-Mb/s Fast Ethernet network interfaces. Twenty-four such Server-Blades mount into a chassis to form a Bladed Beowulf called the RLX system Beowulf. This fit into a rack-mountable 3U space (19" in width and 5.25" in height) [45]. The key technology behind the Green Destiny, which allows it to achieve substantial power savings is the use of Transmeta Crusoe processors [46]. The Transmeta TM5600 CPU generates less than 6 Watts at full load, while the Intel P4 processor generates about 75 Watts. As a result, Transmetas can be packed closely together with no active cooling, resulting in a tremendous saving in the total cost of ownership with respect to reliability, electrical usage, cooling requirements, and space usage [45].

Chapter 3

System Model

3.1 Overview

This chapter describes the organization of power-aware parallel I/O system and the various assumptions made in its design. It also develops an analytical model of the system and quantitatively evaluates the effects of the various parameters. The analytical model leads to a rich space of design options with different tradeoffs. The design space of the system is then characterized and impact of the design choices are evaluated.

3.2 System Description

Parallel file systems typically employ a combination of striping and mirroring in order to evenly distribute the load and to better handle large requests. Our system also employs these techniques. Striping can provide parallelism within a single request (intra-request) while mirroring provides parallelism across requests (inter-request).

In our system, disks are organized into independent groups, each of which contains an entire replica of the file and is self-contained. All groups are assumed to be identical, and the replicas are striped across the disks of a group in an identical

fashion. A single request cannot be split across groups, and once a request is sent to a group, it must be satisfied by that group in its entirety. This is done for reasons of locality and to avoid the overhead of merging sub-requests across groups.

All requests to the system are assumed to be read-only, so replicas need not be synchronized to keep the data consistent across them. Also, requests are first sent to a single front-end module, which may reside either at the client or at the server. This module is responsible for sending the request to an appropriate group, using some load-balancing strategy. If all groups are currently busy serving other requests, the request is queued at the front-end, till some group becomes free. The front-end is not a bottleneck, as its only task is to send the request to the appropriate group, and incurs very little processing overhead. The queue size is also assumed to be very large, to prevent it from being a bottleneck in the system.

Once the request is sent to a group by the front-end, it may access more than one disk in the group. In this case, the request is split into smaller sub-requests, the total number of which is an integral multiple of the stripe size. All the sub-requests which need to be sent to a single disk are aggregated and sent together, so as to aid in better disk arm scheduling and so that the network latency is incurred only once.

In parallel applications, I/O often occurs in bursts [24]. Parallel file systems are typically over-provisioned for the average case, so as to deal with the bursts. As a result, the system can spin down disks during periods of low activity to save power and spin them up again during the bursts, to achieve peak performance. Currently, high performance systems are built from server-class disks which cannot be spun up/down on demand. However, mobile platforms like laptops routinely employ disks with this capability, and we assume that the disks employed by our system have this capability also.

Because a request cannot be split across groups, all the disks of a group could be needed to satisfy any arbitrary request. This means that if some of the disks in

a group were spun down, they may need to be spun up again, in order to satisfy a request. This would place the spin-up delay (of a few seconds) in the critical path of a request, and would not be acceptable for a parallel file system.

To avoid this problem, all the disks of a group are simultaneously spun up or spun down. In other words, the spin-up/spin-down techniques operate at the granularity of a group, rather than single disks. When system demand declines, whole groups are spun down, and when demand rises, whole groups are spun up. To guarantee that no request is made to wait an inordinate period of time, there is at least one group which is constantly spun up, even during periods of low or no system activity.

3.3 Qualitative Analysis

There are two main questions that must be answered: what must be the size of each group and when should they be powered up/down. These two questions are dependent on the amount of performance the system designer is willing to trade off for power conservation. The power-performance tradeoff is expressed as a product of power consumed and the response time. The system designer can supply the maximum power-time product for the system, and the system must make sure that this value is never exceeded, while at the same time, keeping the power consumed minimum.

The number of disks in a group is defined as the size of the group. The groups size forms an upper bound on the number of disks the file can be striped across. It also affects the granularity of power management as smaller sized groups allow more fine grained tuning. Also, at least one group is powered up at all times, so the group size affects the baseline power consumption at near zero arrival rates.

The problem of determining the thresholds in the arrival rate and request length at which to power up/power down groups is a function of both the group size and the amount of performance which can be traded off for a given power saving. The

Symbol	Explanation	Units
n	Number of disks powered up	disks
m	Group Size	disks
D	Stripe Width	disks
l	Request Length	blocks
λ_m	Request Arrival Rate for a Group	requests/sec
λ_n	Request Arrival Rate at Front-end	requests/sec
f	Disk Latency	milliseconds
c	Full Rotation Time	milliseconds
k	Network Latency	milliseconds
h	Software Overhead	milliseconds
i	Network and Interface Overhead	milliseconds
t	Track Size	blocks

Table 3.1: Parameters in the Model

larger the size of the group, the more power needed to power it up and greater the threshold (for a given power/performance constraint). Similarly, for a given group size, the more the performance loss the administrator is willing to tolerate off for saving power, the greater is the threshold (as more delays can be tolerated).

Thus, there is a complex interplay among the group size, request characteristics and the power performance tradeoff. The next section develops an analytical model of these factors and quantitatively studies the impact of varying one or more of these factors.

3.4 Quantitative Analysis

In this section, an analytical model of the system is developed and expressions for power and performance are derived. The analytical model closely follows the one developed in [18].

Assume that the disks in the system are divided into equal sized groups of size m each. Requests arrive at the front-end at a rate λ_n . There are a total of n nodes in the cluster which are powered up, and these are divided into n/m groups. Without

loss of generality, n can be assumed to be an integral multiple of m , as the groups have the same number of disks each. Powering up or powering down a group increases or decreases n by m respectively.

Now, the request arrival rate at each group is (assuming uniform load balancing):

$$\lambda_m = \lambda_n * m/n \tag{3.1}$$

The stripe width of a request is the number of disks the request is split across. The stripe depth of a request is the number of blocks per disk accessed by the request. Let the mean request size be l blocks and the stripe width is D . Then the stripe depth is $\frac{l}{D}$.

The arrival rate for each disk, assuming uniform distribution among the disks (there are no access hot spots) is

$$\lambda_d = \lambda_m \left(\frac{D}{m} \right) \tag{3.2}$$

In general, striping provides two advantages. The first is that it balances the load among disks and avoids access hot-spots. The second advantage is that it allows parallelism for large requests and reduces their service time (up to a point). In this system, load balancing is anyway done at the front-end module. So the main advantage of striping is the parallelism it provides for large requests.

The group size is an upper bound on the stripe width D , which is the parallelism due to striping. In this analysis, the stripe width D is assumed to be equal to the group size m , so the values derived are an upper bound on the performance.

$$\lambda_d = \lambda_n \left(\frac{D}{n} \right) \tag{3.3}$$

Note that D/n is simply the inverse of the number of groups powered up.

Let f , c , k be the disk access latency, time for a full disk rotation and network latency respectively. t is the size of a track in blocks, and i is the network delay to transfer a block of data. h is the software overhead of splitting up a request into D sub-requests.

The disk access latency, network latency and the disk rotational latency are assumed to have a uniform distribution. The mean service time and variation in the service time for a disk [18] are:

$$S_d = \frac{(f + c + k)}{2} + hD + \frac{l}{D} \left(\frac{c}{t} + i \right) \quad (3.4)$$

$$\sigma_{S_d}^2 = \frac{(f^2 + c^2 + k^2)}{12}. \quad (3.5)$$

For tractability, the requests are assumed to follow the Poisson arrival process and hence the inter-arrival times will be exponentially distributed. Each disk can now be assumed to be a server with its own queue, servicing the sub-requests sent to it. As a result, it can modeled as an M/G/1 queue and the queuing delay and response times can be calculated as follows [18].

$$W_d = \frac{\lambda_d(\sigma_{S_d}^2 + S_d^2)}{2(1 - \lambda_d S_d)} \quad (3.6)$$

$$R_d = W_d + S_d = \frac{\lambda_d(\sigma_{S_d}^2 + S_d^2)}{2(1 - \lambda_d S_d)} + S_d \quad (3.7)$$

This is the response time for a single disk. The total response time is the maximum of each of the sub-request times. As shown again in [18], the requests are uniformly distributed among the disks, so the maximum of the response times is given by:

$$R_{max_d} = R_d \left(\frac{2D}{D + 1} \right)$$

For calculating the power consumption, a simplified model of the disk is considered. In this model, a disk has only two states: powered-up and powered-down. The

Symbol	Parameter	Value
f	Disk Latency	18ms
c	Full Rotation Time	8.4ms
k	Network Latency	30ms
h	Software Overhead	1ms
i	Network and Interface Overhead	0.5ms
t	Track Size	22
P_{spin}	Power consumed when disk is spinning	320 mW

Table 3.2: Values used for Analysis

power consumed by a disk in the powered-down (spun down) state is assumed to be 0 Watts and the power consumed in the powered-up (spinning) state is assumed to be P_{spin} watts. The power consumed during the disk access is not modeled as it is negligible compared to the power required to keep it spinning. Hence, the power consumed is directly proportional to the number of disks powered up which is n .

$$P_{total} = P_{spin} * n \quad (3.8)$$

Using equations (8) and (7) the response time - power product can be computed for different values of the group size, request rate and request length. This can help determine the best configuration of the system in order to obtain a desired response time - power product. This is discussed in the next section.

3.5 Discussion

In this section, the design space of the system is characterized using the analytical model derived in the previous section. For the sake of uniformity, the same set of parameter values is used for all the scenarios considered in this section. These are given in Table 3.2 and are identical to the ones used in [18] (except for P_{spin} which does not appear in their model). The value of P_{spin} is the same as that used by [29].

The remaining four parameters in the model are:

1. Request Rate(λ_n)
2. Request length (l)
3. Number of replicas spinning (m)
4. Group Size (D)

Of these, the first two parameters, namely request rate (λ_d) and length (l), are dependent on the input. For a given arrival rate and average request length, the number of spinning replicas (m) needed to achieve a certain response-time power product is studied. This study is carried out for different configurations of the system in which the group size (D) varies from 4 disks/group to 32 disks/group.

It is also assumed that there a total of ten groups or replicas and each can be spun up/down independent of the others. In reality, it may not be possible to realize all these configurations. The total size of the file divided by the storage capacity of an individual disk forms a lower bound on the number of disks/group. Similarly, the total number of disks in the system divided by the group size is an upper bound on the maximum number of spinning replicas.

These physical limitations are not taken into account in the graphs in this section, as this is a theoretical analysis. This evaluates the impact of the design choices on the system, irrespective of physical limitations. However, the physical constraints can be incorporated into the graphs with little difficulty, as they merely prune the design space of the system, and do not change its nature.

3.5.1 Request Rate Versus Number of Replicas

Figures 3.1 to 3.4 show the variation of the response time - power product with arrival rate and number of replicas(groups) which are spinning. Each figure shows

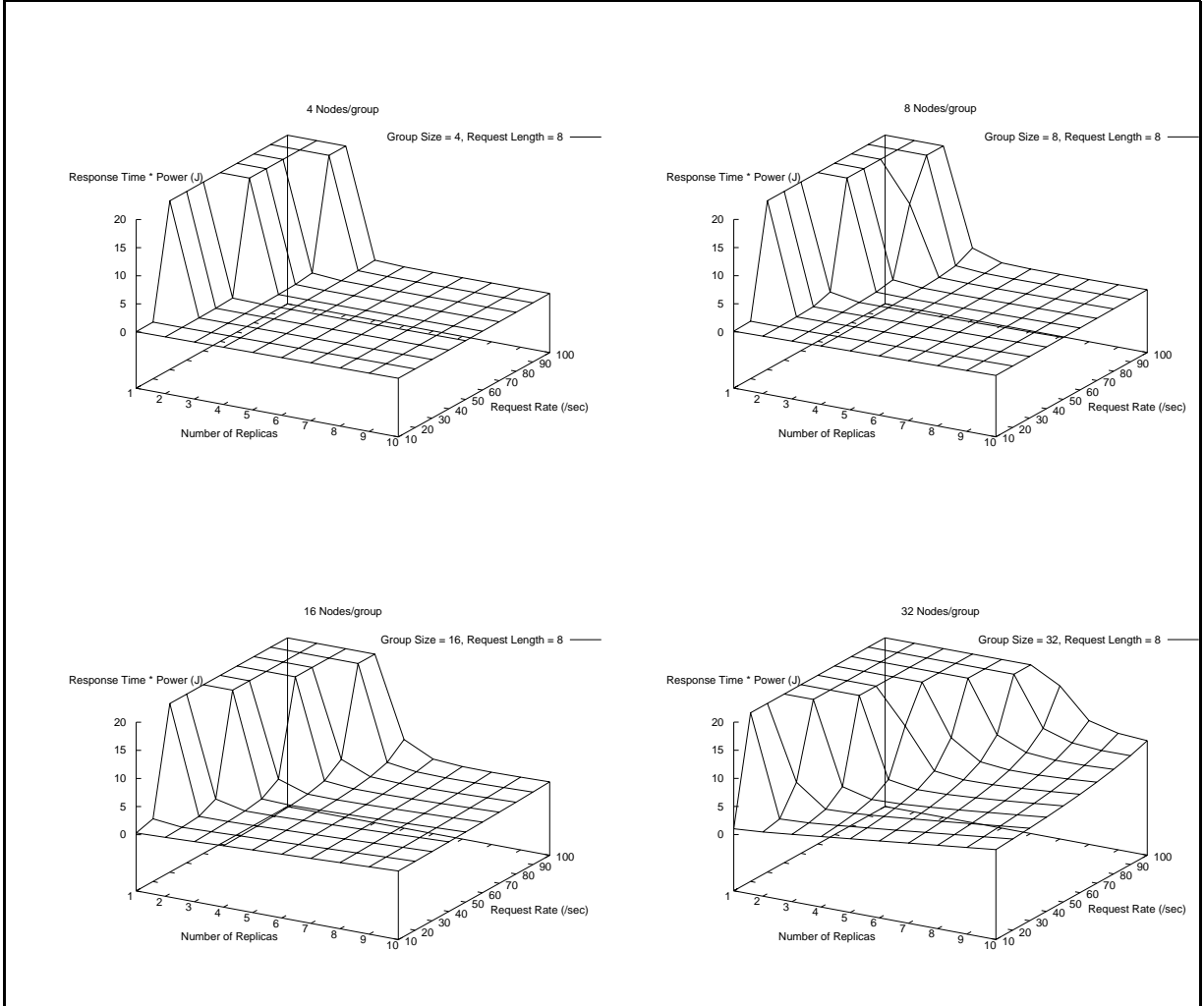


Figure 3.1: Request Rate Vs Number of Replicas for a request size of 8 blocks

the behavior of four system configurations, as described before. The behavior of the system is also dependent on the average request length. Figures 3.1 to 3.4 consider the average request length to be 8, 64, 256 and 1024 disk blocks respectively, thus spanning a wide range of request sizes. If the system becomes unstable, or if its response time - power product exceeds 20 J, the graphs are capped.

For a given request size and a given configuration of the system (disks/group), the response-time power product varies both with the arrival rate as well with as the number of replicas spinning. For a given request rate, there is an optimum value for the number of replicas for which the response time - power product is minimum. This

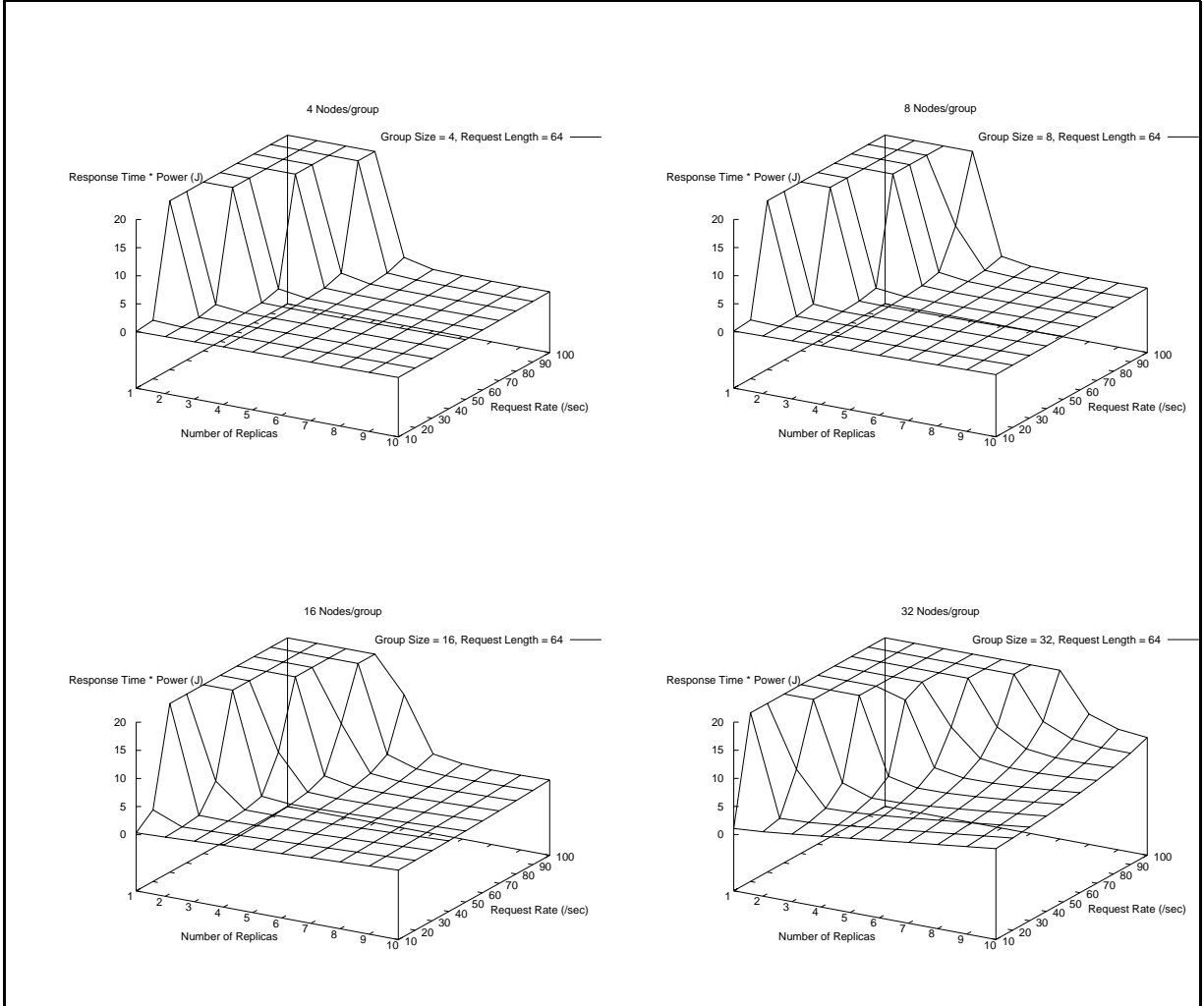


Figure 3.2: Request Rate Vs Number of Replicas for a request size of 64 blocks

can be explained as follows. As the number of replicas decreases, the average request rate at each replica increases (since $\lambda_m = \lambda_n/m$, and m decreases). This corresponds to an increase in the queuing delay at the disks, and increases the total response time of the system. However, the energy consumed goes down as the number of replicas spinning decreases. These are two conflicting effects on the response time - power product.

If the number of replicas spinning is less than the optimum number of replicas, the queuing delay dominates, and there is a sharp rise in system response time, which manifests itself as the sharp increase in the left portion of the graph. If the number

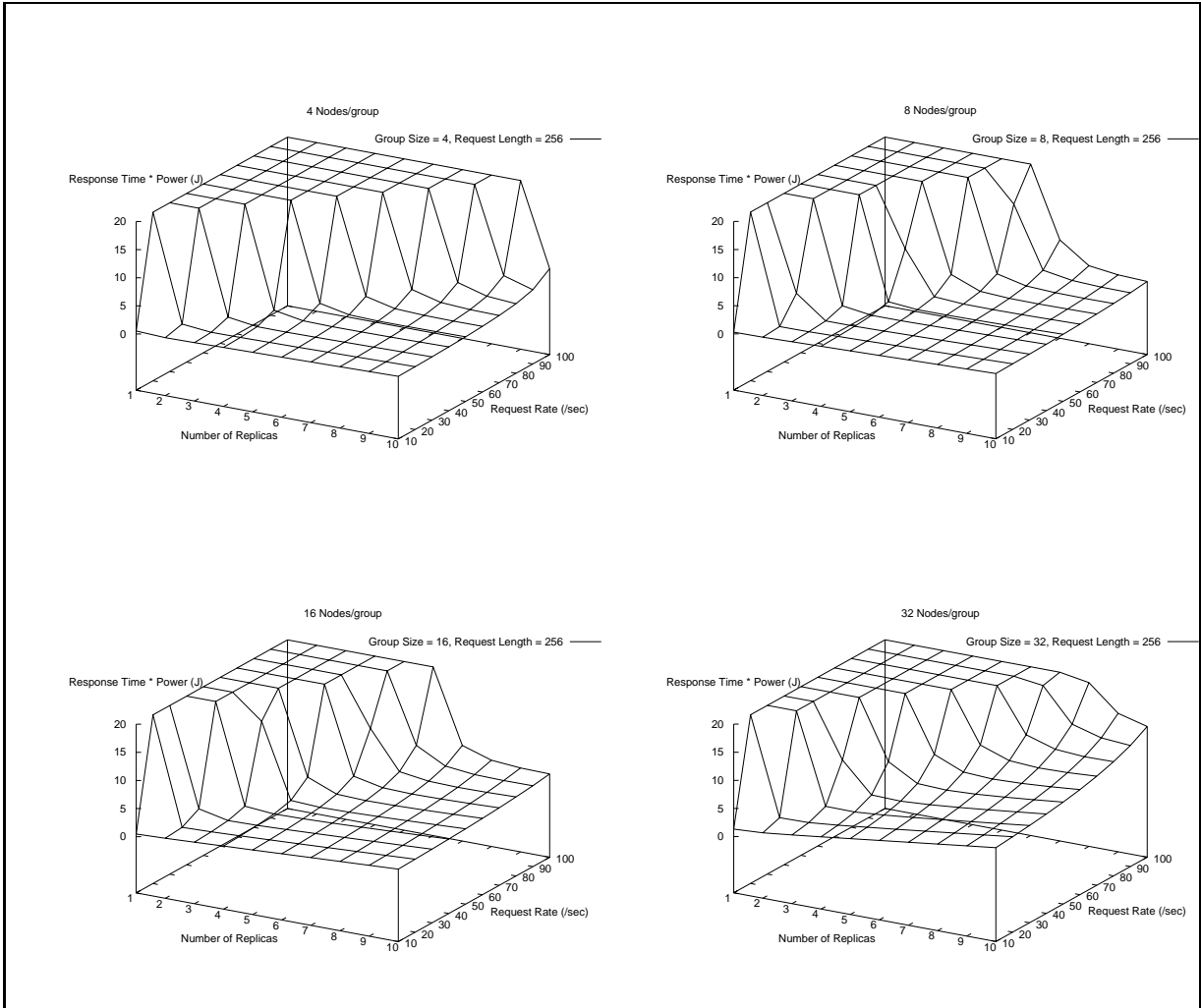


Figure 3.3: Request Rate Vs Number of Replicas for a request size of 256 blocks

of replicas spinning is more than this optimal value, it does not impact the response time much. However, there is a gradual increase in the power consumed, and the response time - power curve gradually increases in the right portion of this graph. To save the maximum power without significant performance degradation, the system must operate as close to the optimal point as possible.

Spinning up more replicas than the optimal number does not affect the response time - power product much. However, having a lower number of replicas spinning than the optimum, causes the response-time power product to rise sharply (it may even make the system unstable). Another thing to be observed is that the optimal

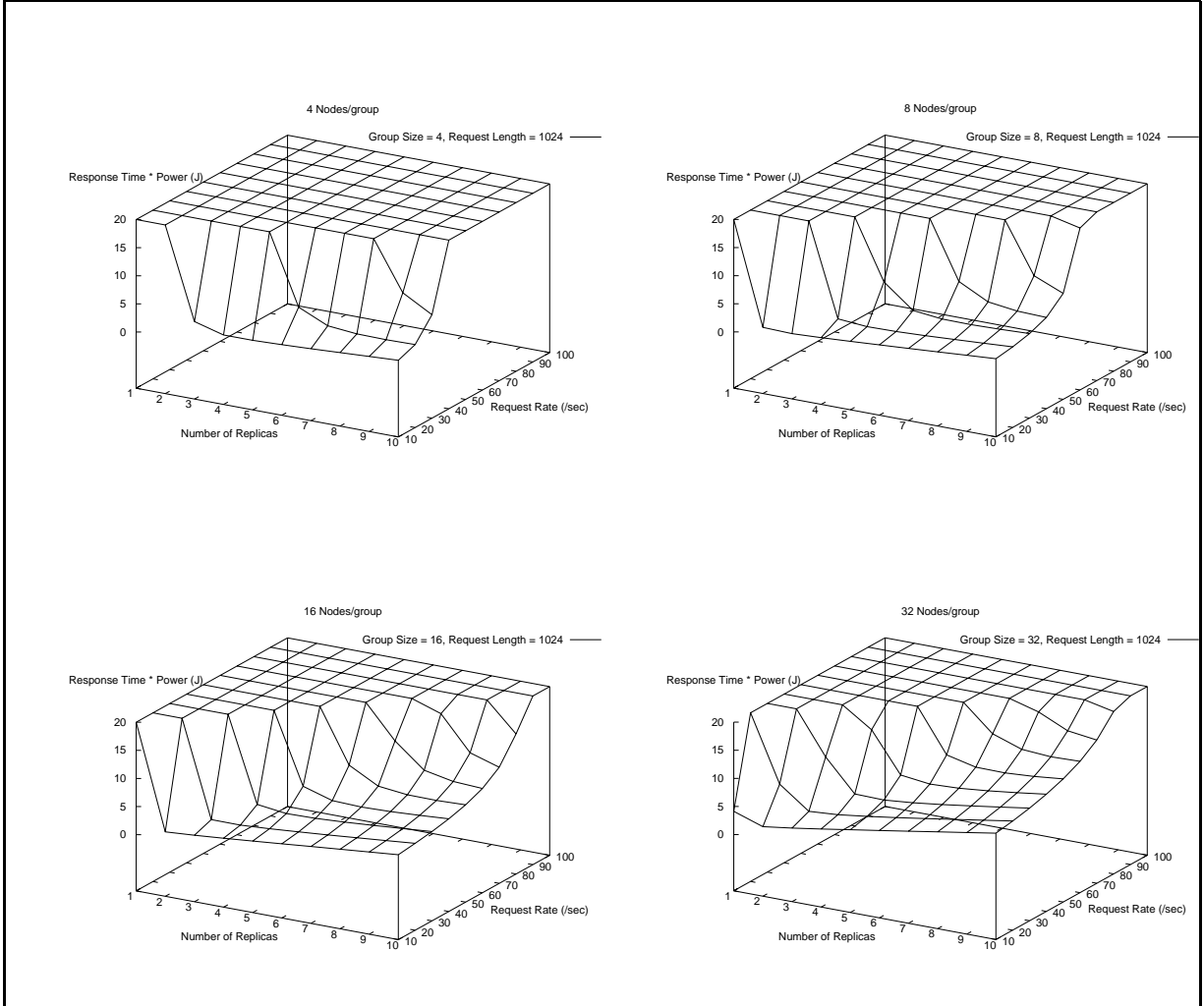


Figure 3.4: Request Rate Vs Number of Replicas for a request size of 1024 blocks

number of replicas spinning, increases as the request rate increases. The reason for this is that the request is likely to expect a greater queuing delay with increase in the request rate. This in turn, could spike up the response time, unless more replicas are spun up to handle it.

Next, the variation of the response time - power product across different configurations of the system is studied. First, the behavior of the system at for small and medium request sizes (figures 3.1, 3.2 and 3.3) is considered. For a given number of replicas spinning and a given request rate, the response time - energy product is more for configurations with larger number of disks/group, as these configurations

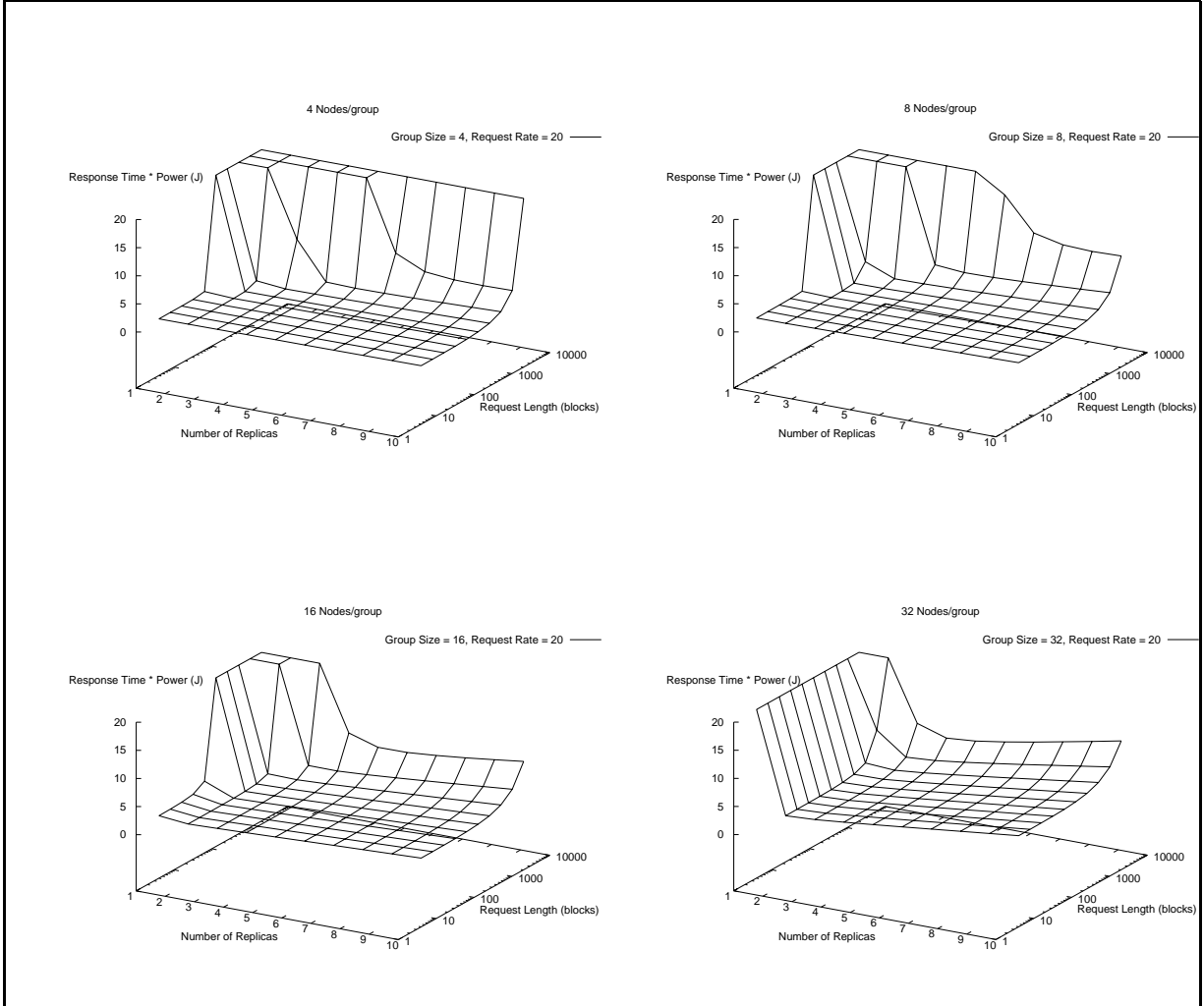


Figure 3.5: Request Length Vs number of replicas an arrival rate of 20 requests/sec

consume more energy.

However, when the request size is very large, the configuration with a greater number of disks/group is able to satisfy the response time - energy constraint up to a larger value of the request rate, than a system with a lower number of disks/group. This is so because the system with a greater number of disks/group has the file striped over more disks, and hence the request can take advantage of greater parallelization than in the system with a lower number of disks/group. As a result, the service time for each individual request goes down, thereby reducing the queuing delay and the total response time. For a system with the smaller number of disks, the queuing delay

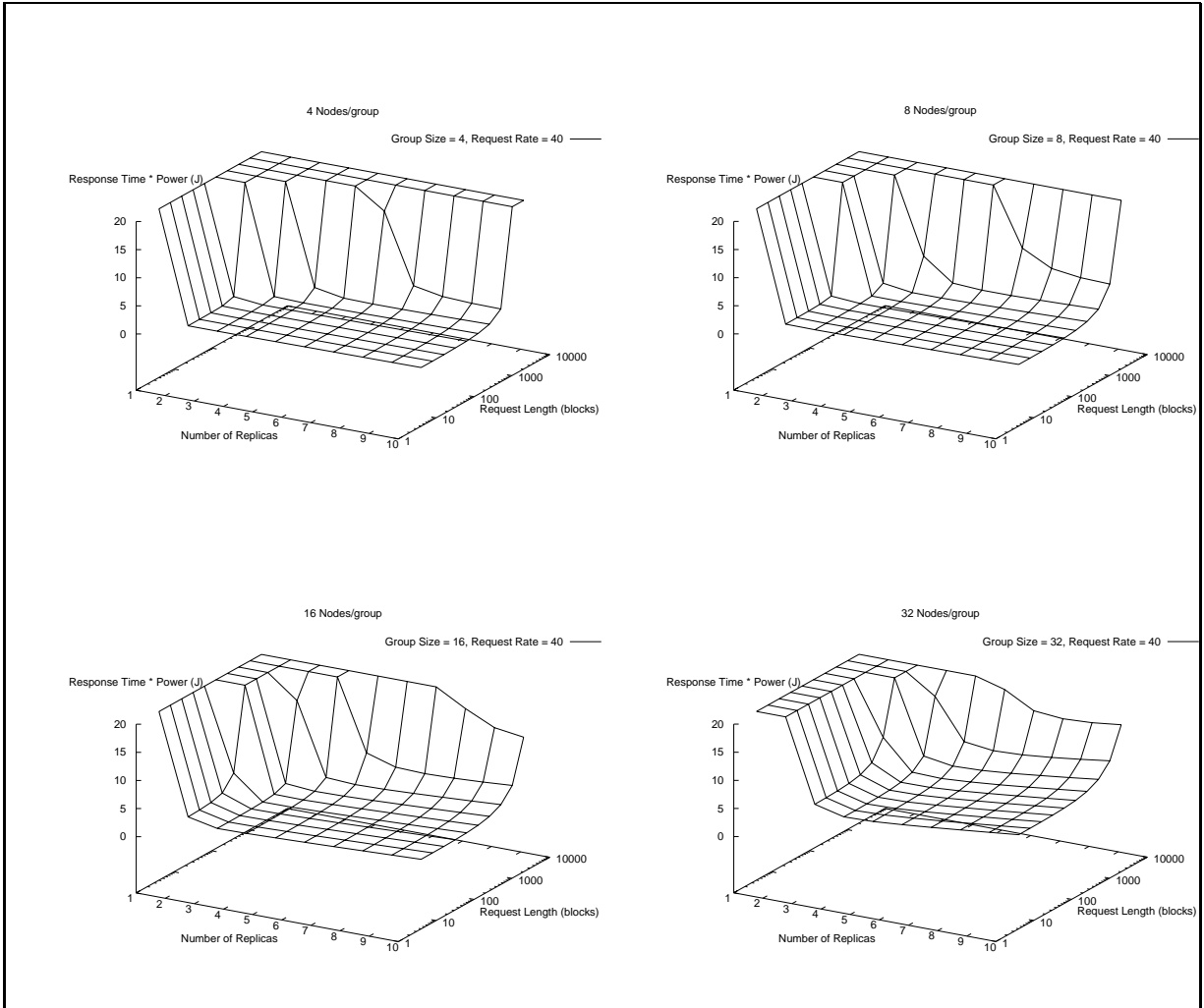


Figure 3.6: Request Length Vs Number of Replicas for an arrival rate of 40 requests/sec

dominates and the system becomes saturated even at lower values of the arrival rate than for the system with the higher number of disks/group.

Thus large requests favor a system with more disks/group as, here, the response time dominates, while small and medium requests favor a system with lesser disks/group, as here, the power consumed dominates.

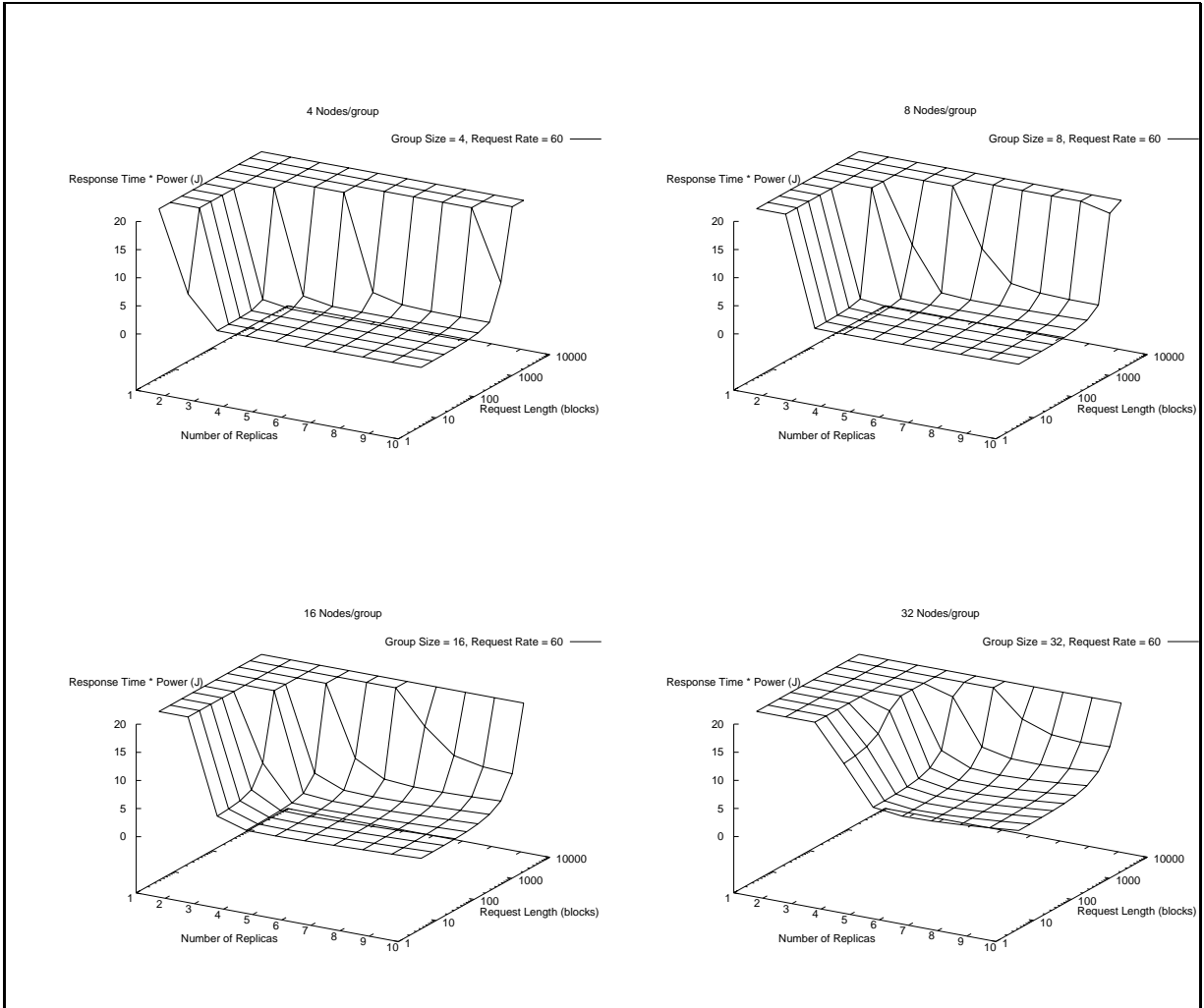


Figure 3.7: Request Length Vs Number of Replicas for an arrival rate of 60 requests/sec

3.5.2 Request Length Versus Number of Replicas

Figures 3.5 to 3.8 show the variation of the response time - power product with average request length and number of replicas(groups) spinning. Each figure shows the behavior of four system configurations, as described before. The behavior of the system is also dependent on the request arrival rate. Figures 3.5 to 3.8 consider the request rate to be 20, 40, 60 and 80 requests/second respectively, thus spanning a wide range of arrival rates. If the system becomes unstable, or if it's response time - power product exceeds 20 J, the graphs are capped.

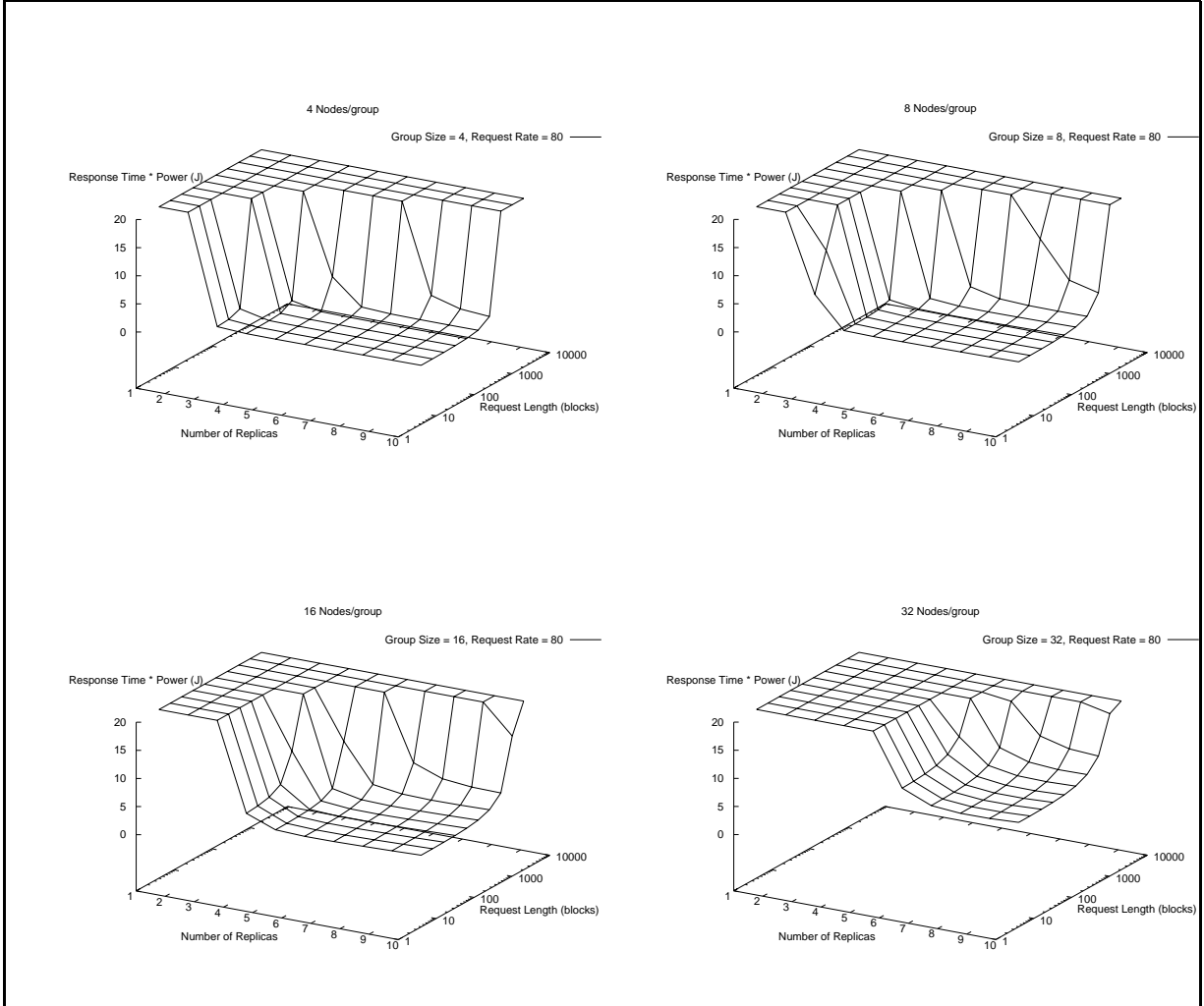


Figure 3.8: Request Length Vs Number of Replicas for an arrival rate of 80 requests/sec

The behavior of the request length curve is similar to the request rate curve. For a given system configuration and request rate, there is an optimum number of replicas for which the response time-energy product is minimized. If the number of replicas is lesser than this optimum value, the response time dominates due to the queuing delay and the product rises sharply with decrease in the number of replicas. If the number of replicas is greater this optimum value, the power consumed dominates and the product increases with increase in the number of replicas spinning, but more gradually. Again, as the request length increases, for a given request rate, the optimum number of replicas for a given system configuration also increases.

When comparing system configurations at different request rates, the opposite effect to what was described for the request rate vs number of replicas graphs is observed. At lower request rates, a system which has a greater number of disks/group is more efficient for large requests than a system with a lesser number of disks/group. The reason is that at low arrival rates, the dominant factor in response time is the service time of a request. For large requests, this can be improved considerably by striping across a greater number of disks and hence enabling greater parallelism. However, it is possible to do this only if the group has enough disks to stripe across. For configurations with a smaller number of disks/group, the response time of the system becomes very high due to the large service time component, and the response time - power product rises steeply. This is especially true if the number of replicas spinning is low, as requests are now forced to wait in the queue for requests before them to complete and increasing service time can increase the queuing delay of these requests as well, and the effect is compounded.

Chapter 4

Implementation

4.1 Overview

The last chapter introduced the system model and evaluated the various trade-offs in its design. In this chapter, we will describe the implementation of the system and the simplifications made. We will first describe the architecture of the system and its various components. We will then describe some of the key algorithms used in the system.

4.2 Architecture

The system can be broadly divided into four major sub-systems. The *Request-Generation Sub-System* is responsible for generating requests according to certain distributions or from a real application trace. The *Topology and Request-Routing Sub-System* is responsible for implementing the topology of the system described in the previous chapter and for sending requests through the topology. It is also responsible for carrying out the commands issued by the *Adaptation Sub-System* to spin disks up or down. The *Adaptation Sub-System* is responsible for deciding when the system should adapt and by how much it should adapt. Finally, the *Statistics Gathering and*

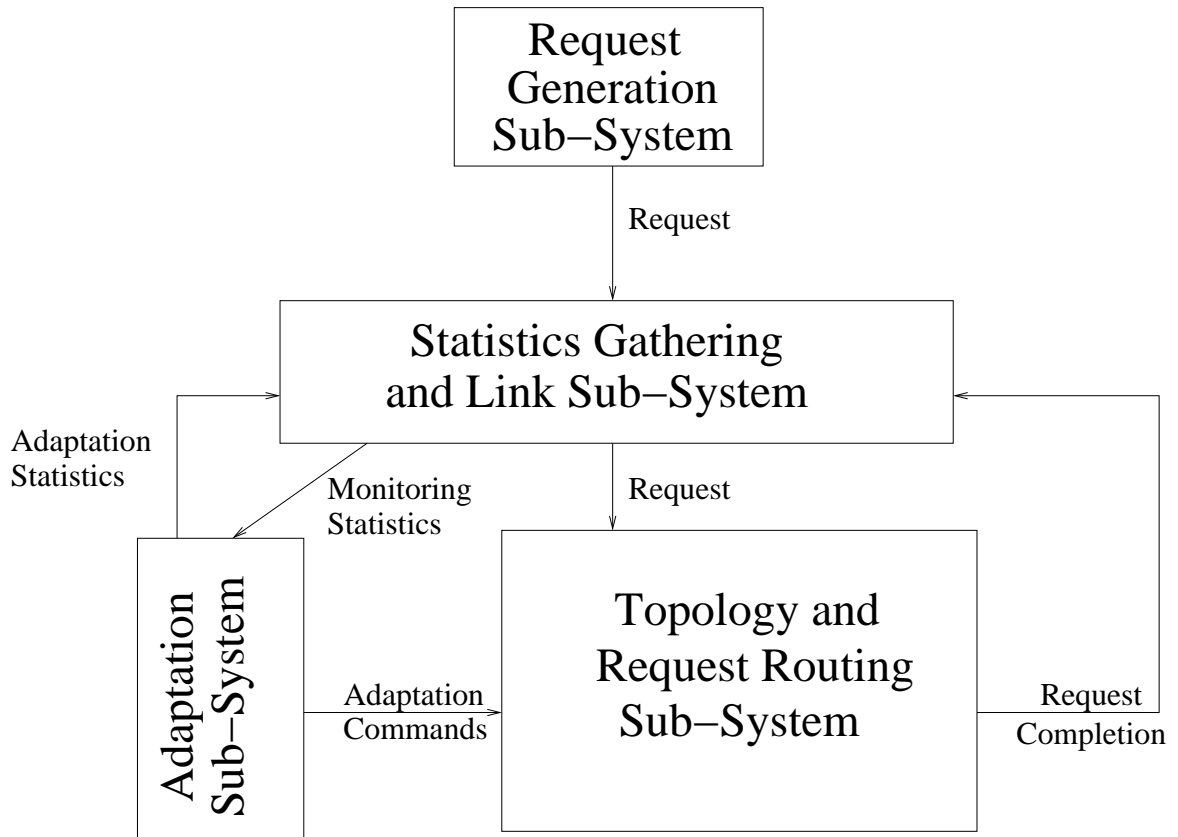


Figure 4.1: The various sub-systems and their interactions

Link Sub-System orchestrates the interactions between the various sub-system and gathers statistics of importance from them.

This section describes the various sub-systems present in the system and their interaction. This is shown in figure 4.1.

4.2.1 Request-Generation Sub-System

This sub-system is responsible for generating requests according to some specified distribution or from traces gathered from real applications. There are three parameters which can be specified:

- Request inter-arrival Times

- Average Request Length
- Offset of the Request in the File

The first two parameters can be generated either from a distribution or from an application trace. The third parameter is not of importance to our study and is simply chosen in a linear incremental fashion, that is, a new request begins where the last request left off. This models a sequential access pattern, but since the system don't do any prefetching or caching, it doesn't provide any additional benefit.

The request inter-arrival times can be specified from any of the following distributions:

1. Exponential Distribution
2. Bursty-Exponential Distribution
3. b-Model

The exponential distribution is used for validating the analytical model for the system in section 3.4. The Bursty-Exponential distribution is obtained by combining the exponential distribution with an ON/OFF distribution. The b-Model distribution is used to generate bursty, irregular traffic and is based on the paper by Wang et. al.[47]. It is described in detail in section 4.3.

The request rate can also be specified from an application trace. Here again, there are two possibilities: the trace can be played back asynchronously or in a synchronous fashion. In the first case, requests are issued at the times specified in the trace file, irrespective of the time taken by the system to service previous requests. In synchronous mode, the next request is issued only if the previous request is completed by the system. The arrival times in the trace are adjusted taking into account the duration of the request in the real application.

The request length can be specified from either a uniform distribution or an application trace. It is also possible to specify more complex distributions such as Zipf and hyper-exponential, though these aren't supported yet.

4.2.2 Topology and Request-Routing Sub-System

This sub-system models the organization of the system and routes requests through it. All requests are sent to a single, central component called the *balancer*. The balancer then sends the request to a particular group, which will then satisfy the request. Each group has a replica of the file striped across its disks, and the request may need to be split into multiple sub-requests for each disk. This is done by the *splitter* component. Each group has a *splitter* associated with it, and all requests to a group are sent to that group's splitter by the balancer. The splitter then splits the request and routes the individual sub-requests to the disks of the group. It also recombines data from each sub-request after it has been satisfied and sends the data back to the client. Figure 4.2 shows the organization of the Topology and Request-Routing sub-system.

Balancer Component

This component is responsible for routing a request to a group from the list of active groups (spun-up groups). This ensures that the request is not kept waiting for a group to spin up before it can be satisfied. In case, all groups are busy, the balancer queues the request till a group becomes idle. Thus the balancer makes sure that only one request is sent to a group at a time. This limits the concurrency of requests in the system to an extent, but ensures that there is no queuing of the requests at the disks. This makes it easier for the system to provide guarantees on the response time of a request, and also simplifies the adaptation mechanism (discussed in next section).

An alternate implementation of the balancer is to send the requests to their chosen group as and when they arrive, and queue them at the splitter, instead of at the

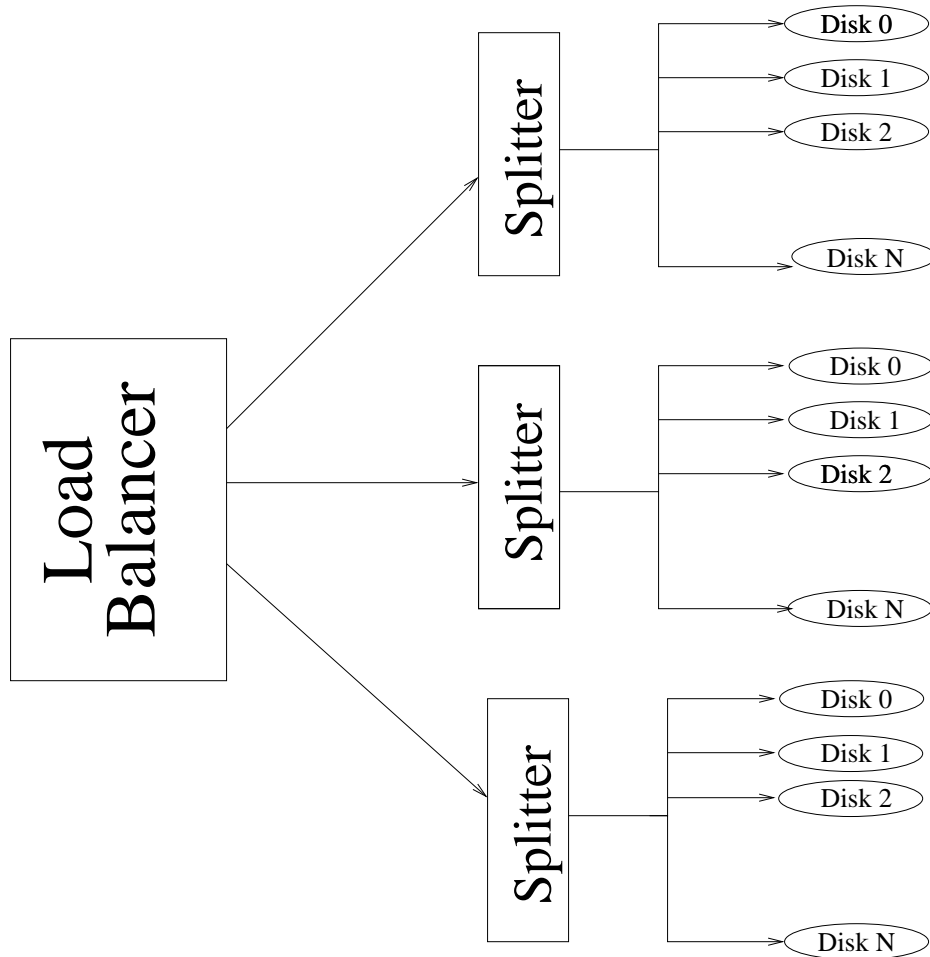


Figure 4.2: Block Diagram of the Topology and Request-Routing Sub-system

balancer. But from a queuing theory perspective, it is more efficient to have one central queue than have multiple independent queues and load balance between them. Having a queue at each splitter would also complicate the adaptation mechanism. Before a group can be spun down, it must either satisfy all the requests queued at its splitter, or the requests must be sent to other splitters' queues. The first option would make the system slow in responding to a command from the adaptation mechanism. The second phenomenon is known as *jockeying* in queuing theory and is difficult to model. Taking these difficulties into account, we use a centralized queue for all

requests in the system. On the other hand, having multiple, distributed queues will allow us to overlap the network latencies between the balancer and the splitter, and may be a better option if the groups are at geographically distributed locations.

Splitter Component

The splitter is responsible for splitting a request into multiple sub-requests and sending each sub-request to the disk it is intended for. Since the data sought by a request may be striped across multiple disks, it is the splitter's responsibility to ensure that the right data is requested from the right disk. The size of each sub-request is dependent on the stripe size and the size of the original request. The algorithm for generating sub-requests and sending them to their respective disks is discussed in section 4.3.2. After sending the sub-requests, the splitter waits for each disk to satisfy its sub-request. It then recombines the data from the disks and sends it back to the client which requested it. Only after all these steps are completed, does it start processing the next request. Though it is possible to overlap the issue of the next request with the recombination of the current request, this is not implemented in the interest of simplicity. Moreover, since the recombining overhead is usually very small, this is not a significant bottleneck in the system, except at high loads.

Disk Component

This models the disk described in the analytical model in section 3.4. There is a certain latency to access the disk for the first time, and a delay to transfer the data. The latency is fixed or follows a uniform distribution, while the delay is proportional to the size of the data. We also assume that the network delay and latency is incurred at the disk only. Though there is a latency and delay in transferring the data from the balancer to the splitter, this is not modeled in the system, as the splitter is an artifact of our implementation and not a part of the system model. A real implementation

would combine the functions of the balancer and the splitter into a single module.

4.2.3 Adaptation Sub-System

The adaptation sub-system decides when the system should adapt and by how much it should adapt. In particular, it decides how often groups must be spun up/down and how many groups should be spun up/down at a time. This decides how quickly the system responds to changes in the workload.

The adaptation sub-system has two components: *Monitor* and *Controller*. The monitor component decides how often the system should adapt to a change in the load and the controller component decides the degree of adaptation, that is, how much should it adapt.

Monitor Component

The monitor component is a passive component which observes changes in system load over a period of time, and decides when the system should adapt to those changes. If the time interval of adaptation is too long, the system can either run at a sub-optimal configuration or find itself unable to respond to the incoming workload. The first case merely wastes energy and resources, while the second case may result in the system being unable to satisfy the extra load, in spite of having the resources to do so. If the time interval of adaptation is too small, then the system may be over-responsive to transients and jitters in the workload. It may end up wasting energy constantly spinning disks up and down, and never attain a stable state. For these reasons, it is very important to choose the right interval of adaptation in the monitor.

The monitor component also decides what metric to observe in the system. It must choose a metric which is indicative of changes in the input load, but doesn't fluctuate too wildly for small changes in the input. The choice of the metric also reflects what factor is most important to the user like energy, throughput, response time etc. For

simplicity, we assume that the monitor observes only one metric, and that there is only one monitor in the system. This ensures that multiple monitors don't clash with each other, and that the system responds only at -determined instants of time. Currently, either the average queue length, average response time or the average utilization of the system can be monitored. The monitor algorithm is discussed in section 4.3.3.

Controller Component

Once the monitor has decided when to adapt, it alerts the controller component. The controller component decides how much to adapt, based on the value reported by the monitor. It then translates the adaptation decision into a series of commands which are then issued to the Topology and Request-Routing sub-system.

Based on the value reported by the monitor and other considerations, the controller can choose one of three courses of action. It can choose not to adapt at all, to spin up a certain number of groups or to spin down a certain number of groups. Currently, the only controller supported is a *MinMax* controller and its operation is discussed in section 4.3.3.

4.2.4 Statistics Gathering and Link Sub-Systems

This sub-system orchestrates the interactions between the various sub-systems and collects statistics from each of them. It consists of two components: the *Front-end* component and the *Statistics* component. The former is used to route messages between the various sub-systems, while the latter is used to gather statistics from different sub-systems and present them in a unified fashion to the user.

Front-end Component

The front-end component serves a central message board for the other sub-systems. It is responsible for routing both requests and commands to the various sub-systems.

The request-generation sub-system submits the requests to the front-end, which are then sent to the Topology and Request-Routing sub-system. Similarly, the adaptation sub-system sends the adaptation commands to the front-end and these are sent to the Topology and Request-Routing sub-system, as well.

Statistics Component

This provides a centralized repository for gathering statistics about the system and presenting them to the user in a unified way. It gathers statistics about the request distribution from the request-generation sub-system and statistics such as utilization and queue length at each component of the Topology and Request-Routing sub-system. It also gathers statistics about the average response time and average queuing delay of each request in the system as a whole. Finally, it gathers statistics from the adaptation sub-system like how often the adaptation was effected and the average value of the monitored parameter. It also keeps track of the average power consumed by the system and the power consumed in spinning disks up and down.

4.3 Algorithms

In this section, we will describe some of the important algorithms used by the components described in the previous section. The algorithms are grouped according to the sub-system which implements them as *Request-Generation Algorithms*, *Request-Routing Algorithms* and *Adaptation Algorithms*.

4.3.1 Request-Generation Algorithms

This section surveys the b-model algorithm, its rationale and implementation. It is based on the paper by Wang et. al [47].

b-Model

The b-model is a method for generating bursty I/O traces, which are representative of real file system workloads. Disk I/O traffic is usually bursty and self-similar and cannot be accurately modeled with Poisson arrivals [48]. Though many models have been proposed to capture burstiness (fractional Brownian motion, ARIFMA), the b-model is unique in that it requires very few parameters to fit and can generate large traces quickly. Also, traces generated with the b-model fit real world traces well in terms of queuing behavior.

The b-model involves only a single parameter, the bias b , which is directly related to the burstiness of the data. The b-model is closely related to the "80/20" law; 80% of the accesses involve 20% of the data. In the b-model, this would correspond to the bias parameter $b = 0.8$. The construction begins with a uniform interval and recursively subdivides the number of accesses to each half, quarter, eighth, etc. according to the bias b . Thus step $n + 1$ ends with a total of $2^{(n+1)}$ data points, which are obtained by splitting each of the 2^n points from step n according to the formula:

$$\begin{aligned} Y_t^{(n+1)}(2 * i) &= Y_t^{(n)}(i) * (1 - b) \\ Y_t^{(n+1)}(2 * i + 1) &= Y_t^{(n)}(i) * b \end{aligned}$$

for $i = 0, 1, \dots, 2^n - 1$ and $b \in [0.5, 1)$. The exponent in $Y_t^{(n)}$ indicates the current step and is also called the aggregation level. The above formulas are for the deterministic version of the model, where the split is always done in the same direction. In a real trace generation, b will go randomly to the left or right to create some randomness in the synthetic trace. Due to the multiplicative, cascading process during the construction, the b-model generates a self-similar trace with high irregularity, which depends on b . The closer b is to 1, the higher the irregularity and $b = 0.5$ gives a

uniform trace.

In order to generate traces from the b-model, we need to specify two other parameters in addition to the bias b . They are, the total number of requests in the trace N (total volume), and the aggregation level n , which determines the number of data points which will be generated, that is, $l = 2^n$. Though, we can generate traces using the above construction, a more efficient way is to use a stack. Initially, the total volume N is pushed on top of the stack. At each step, the algorithm examines the value at the top of the stack. Conceptually, each point is associated with an aggregation level n , which is stored on the stack along with the point. If the aggregation level is n , the algorithm outputs the point. Otherwise, the top data point is split according to the bias b and replaced by two new points of a higher aggregation level. This algorithm is shown below:

1. Initialize the stack and push pair $(0, N)$ onto the stack
2. If the stack is empty, all the 2^n data points have been generated and the process ends.
3. Pop a pair (k, v) from the stack. If $k = n$, output v as the next data point and go back to Step 2.
4. Flip a coin. If heads, push pair $(k + 1, v * b)$ and $(k + 1, v * (1 - b))$ onto the stack. Otherwise, push pair $(k + 1, v * (1 - b))$ and $(k + 1, v * b)$ onto the stack. Go back to Step 2.

The traces generated by the above algorithm vary in burstiness according to the bias b . Figure 4.3 shows the traces generated with $n = 1000$, $N = 10000$ and b ranging from 0.5 to 1.0. Note that $b = 0.5$ yields a smooth trace, while $b = 1$ results in a single large burst.

4.3.2 Request-Routing Algorithms

This section describes the algorithm at the splitter for splitting a request into multiple sub-requests and sending them to the appropriate disks.

Splitter Algorithm

The first thing the splitter does is to compute the number of sub-requests as the request length divided by the stripe size. There are two cases: either this is lesser than the number of disks in a group or it is greater. In the first case, the disks to which the sub-requests must be sent are identified, and the request is split into sub-requests which are then sent to each of these disks. In the second case, each disk may receive more than one sub-request. Since there is a latency every time the disk is accessed, all sub-requests to a disk are aggregated and sent to it, so that the latency is incurred only once. The same holds for the network latency from the splitter to the disk.

If the number of sub-requests is greater than the number of disks in the group, the algorithm proceeds in multiple rounds. For each disk, a list of sub-requests to be sent to the disk is maintained at the splitter, and in each round, the algorithm adds one sub-request to the list for each disk. The length of a sub-request is equal to the stripe size, for all but the first and last sub-requests. The offsets of the sub-requests are computed in an incremental fashion from the offset of the original request. When all the rounds are done, the list of sub-requests for each disk are aggregated and sent to the disks. The splitter then waits for each of the disks to satisfy the sub-request sent to them. When all the data is returned, it is aggregated together and sent back to the client.

4.3.3 Adaptation Algorithms

This section describes the algorithms in the adaptation sub-system viz. the monitor and the controller.

Monitor Algorithm

The monitor algorithm is a tight loop which repeatedly observes some parameter of the system for a period of time. It smoothes the observations over a window of time, so as to eliminate outliers and jitters. At the end of the specified time period, the monitor alerts the controller with the mean value of the parameter observed during the last time-window. After the controller returns, the monitor may decide to throw away the observations made during the previous time-window, or it may retain some portion of it in the next time-window of observation as well. This depends on the shift parameter, which decides how much of overlap is present between successive time-windows. Finally, the sampling frequency specifies how often the parameter value is recorded in the specified time window. Figure 4.4 shows the monitor parameters and the relationship between them.

Thus, there are four parameters which must be specified for the monitor: the *time period of observation*, the *size of the time-window*, the *shift between successive time-windows* and the *sampling frequency*. The time period of observation is the minimum time between successive adaptations of the system, and decides how quickly successive adaptations take place. The size of the time-window decides how much time the parameter is observed before any adaptation decision is taken. This usually smaller than the time period of observation, so that transients introduced due to the last adaptation die down. However, it must be large enough that the system is not influenced by short bursts or jitters in the input workload, and can establish the mean value of the parameter being observed with some confidence. The shift parameter decides how much memory we want the system to retain. The lesser the

shift parameter, the more the system is influenced by observations from the previous time-window, and more the memory in the system. It introduces some hysteresis in the system, and is present to prevent the system from oscillating wildly. However, it must not be too small, as otherwise the system will be biased by adaptation decisions made in the past, and may not adapt adequately to the current workload. The last parameter is the sampling frequency. A high value of the sampling frequency allows the monitor to estimate the parameter under observation accurately, but introduces a large performance overhead on the system. For parameters which do not fluctuate too much, a smaller value of the sampling frequency would suffice to give the required accuracy, while making the monitoring less intrusive.

Controller Algorithm

The only controller currently supported by the system is the *MinMax* controller, which tries to keep the value of the monitored parameter between two fixed bounds *min* and *max*, respectively. If the mean value of the parameter reported by the monitor is within these bounds, the controller takes no action. If the mean value is greater than *max*, it spins up more groups to handle the load. If it is lesser than *min*, it spins down some groups to conserve energy. The number of groups to spin up and spin down in either case, is given by the parameters *upStep* and *downStep*, respectively.

The choice of these four parameters: *min*, *max*, *upStep* and *downStep* influences the responsiveness of the system to changes in the load. The *min* and *max* parameters define the acceptable limits of tolerance of the system. Any value of the parameter between *min* and *max* is considered stable. If the parameter is more than *max*, that means the system is in degraded performance mode, and more disks must be spun up to increase the performance. If the parameter is less than *min*, that means the system is operating at a higher performance configuration than necessary. The system

in said to be in an energy-inefficient mode and disks can be spun down to conserve power. If too many disks are spun down however, the system may end up going to the degraded performance mode. Similarly, when the system is in the degraded performance mode, spinning up too many disks can end up taking the system to the energy-inefficient mode. For this reason, the parameters `upStep` and `downStep` must be chosen carefully depending on the values of the `min` and `max` parameters and how quickly the monitored parameter value changes after the system has been adapted. The modes of operation of the system with respect to the MinMax controller are shown in figure 4.5.

4.4 Summary

This chapter described the implementation of the system introduced in the previous chapter. The architecture of the system was presented and the various sub-systems were introduced. The second part of the chapter dealt with the algorithms employed in the various sub-systems present in the system.

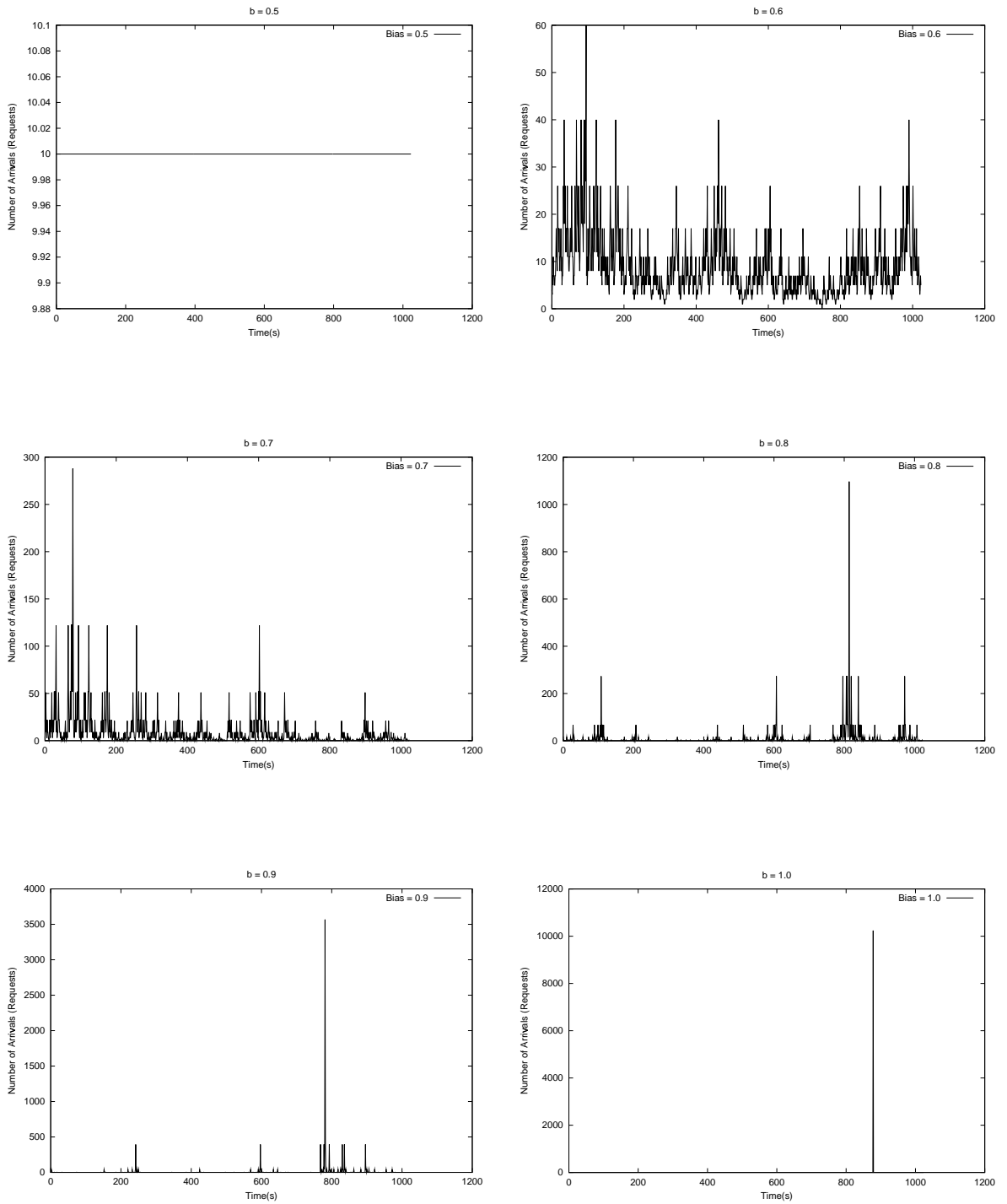


Figure 4.3: Traces generated by the b-model $n = 1000$, $N = 10000$ for different values of b

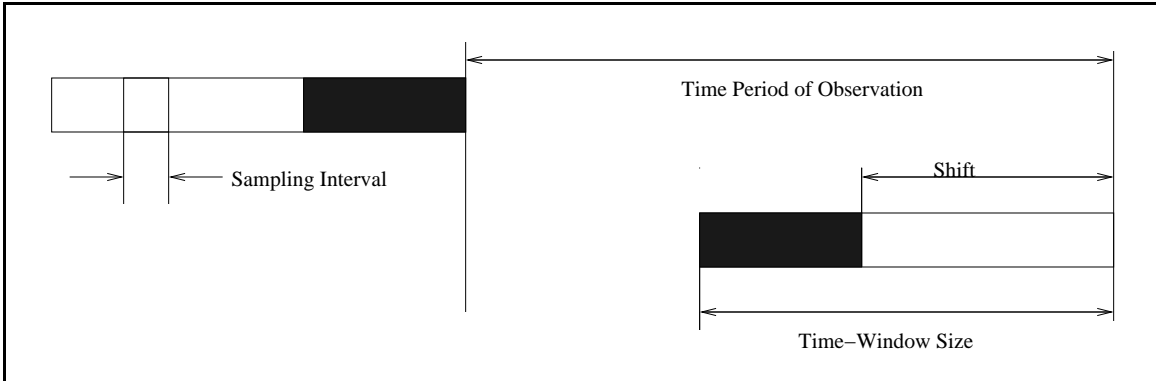


Figure 4.4: Monitor parameters and their relationship

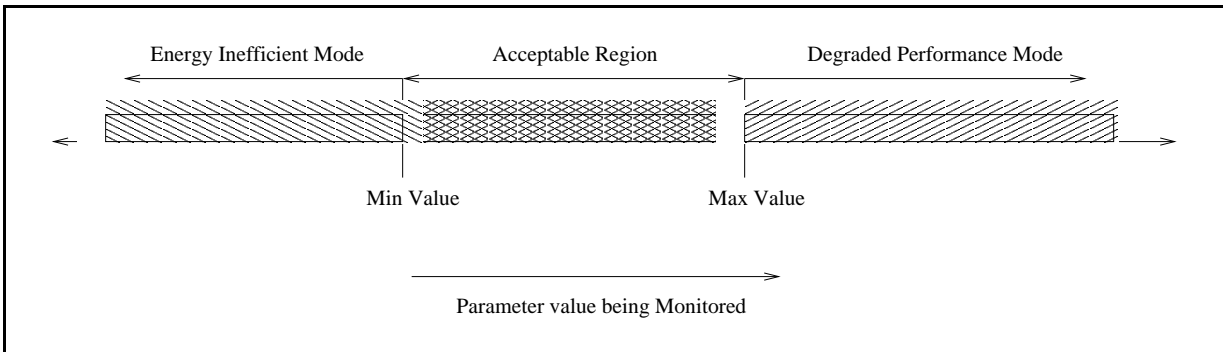


Figure 4.5: Modes of Operation of the System with respect to the MinMax Controller

Chapter 5

Experimental Infrastructure and Results

5.1 Overview

This section discusses the experimental infrastructure we developed to study the system and its components. We present results from running both synthetic and real workloads on the system. The synthetic workloads are generated by the b-model [47] discussed in the previous chapter and the real workloads correspond to two parallel applications discussed in this chapter.

5.2 Experimental Infrastructure

5.2.1 Simulation

To study power management , we developed a simulation of the system using the CSIM+ simulation engine [49]. CSIM+ is a process-oriented discrete-event simulation package for use with C and C++ programs. It provides classes and abstractions for commonly used simulation concepts such as processes, events, facilities and messages.

It also supports common statistics gathering and analysis routines, as also features for automatic run-length control. A more complete description of CSIM+ can be found in [50].

The simulator consists of approximately 3000 lines of object-oriented C++ code and Perl scripts. It is written with flexibility and adaptability in mind. It allows the designer to play with various combinations of policies and design parameters with ease. It can be configured externally with a script file, allowing the system developer to rapidly experiment with various parameter values and study their effects on power and performance. It also allows the user to configure how much data is collected about the various aspects of the system and to represent it in a concise form.

5.2.2 Run Length Control and Transients

One of the most important questions in a discrete-event simulator is when to terminate the simulation. This issue is referred to as *run-length control* by many simulation textbooks [51]. Running a simulation for too short a time may lead to inaccurate results, while running a simulation for too long a time leads to wastage of computing resources. There is also the problem of the initial transient when running a simulation. CSIM simplifies the issue of run-length control by allowing the user to choose the desired accuracy and confidence interval, and choosing the run-length automatically. Nevertheless, there are some complications which arise due to the adaptive nature of the system and the types of workloads used.

The main problem with simulating an adaptive system is that we do not know when the system has adapted to the workload. If we apply the normal algorithms for testing convergence to the simulation, we may find that the system response may have converged to an accurate but non-optimal value as the system has not had time to adapt. Adaptation introduces transients and we must wait for these to die down before reporting the simulation results. The problem is complicated by the fact that

some systems may never converge in the presence of fluctuating workloads, as they keep adapting all the time.

We deal with this problem in the following way: we wait for a minimum interval of time before applying the test for convergence to give the system time to adapt. If it still has not converged by then, we run the system for some more time to give it a chance to converge. We keep repeating this at periodic intervals and checking for convergence. If it still has not converged after a maximum interval of time, we simply report the values obtained. If this maximum interval of time is large enough, then this value will represent the average behavior of the system over a long period of time, and we choose this value as representative of the final value of the system, had it converged. While we do not have any theoretical bounds on this value, our observations over many trials seems to indicate that this is a fairly accurate estimate of the system response. Also, we initially start with all replicas active, and then spin them down gradually depending on the system load.

5.2.3 System Parameters

Table 5.1 shows the default parameter values we use in the simulation. It is divided into various sections for the sake of clarity. Some of these parameter values are the same as the ones we used in the analytical model in section 3.4. This allows us to calibrate the accuracy of the analytical model. The run-length control section of the table shows the parameter values we use to control the accuracy of the simulation.

The disk parameters are from [18] which are based on the WDE-4360 Western Digital Ultra-SCSI hard drive. This is typical of disks used in real servers, but this disk does not support dynamic power management. As a result, we use the spin-up/spin-down values from the from [29] for the Quantum Go-Drive 160. The topology parameters are representative of many real systems and are similar to those in [18], with the following difference: rather than considering that the data is striped across

Name	Value	Units	Comment
<i>Spin-up/ Spin-down Parameters</i>			
SpinUpDelay	1.0	seconds	Time taken to spin-up a disk
SpinDownDelay	0.5	seconds	Time taken to spin-down a disk
SpinUpPower	0.3	Watts	Power consumed in spinning up a disk
SpinDownPower	0.0	Watts	Power consumed in spinning down a disk
SpinningPower	0.3	Watts	Power consumed in keeping a disk spinning
<i>Topology Parameters</i>			
StripeSize	64	Disk blocks	Size of each disk stripe
NumDisks	128	-	Number of disks in system
NumReplicas	8	-	Number of replicas (Groups) in system
ClusterSize	16	-	Number of disks/group
<i>Disk Parameters</i>			
Latency	18	milliseconds	Latency of disk access
RotDelay	8.4	milliseconds	Rotational delay of disk
NetLatency	30	milliseconds	Latency of network
NetDelay	0.5	milliseconds	Network delay for one disk block
TrackSize	22	disk blocks	Size of each disk track
<i>Monitor and Controller Parameters</i>			
SamplingInterval	1	seconds	Interval between observations by monitor
WindowSize	300	seconds	Size of monitor window
Shift	240	seconds	Shift between monitor windows
UpStep	1	-	Number of groups spun-up by controller
DownStep	1	-	Number of groups spun-down by controller
<i>Run-Length Parameters</i>			
Accuracy	0.01	-	Number of significant digits in result
Confidence	99	%	Confidence Interval for simulation
MinTime	10000	seconds	Minimum logical time to run the simulation
MaxTime	100000	seconds	Maximum logical time to run simulation

Table 5.1: Parameter values used in simulation

all the 128 disks in the system, we assume that the system is divided into 8 groups, each of which consist of 16 disks and that a replica of the data is striped across the disks in each group. This is done to suit our system model, so that we can study the effect of varying the group size and the number of disks/group while keeping the total number of disks fixed. The monitor and controller parameters are chosen based on what we think are reasonable values for the adaptation mechanism. We assume that the system samples the metric of interest every 1 second and used a window size of approximately 5 minutes to adapt. We believe that this is a reasonable compromise

between the sensitivity of the system to variations in the workload and its long term reliability, as a system that adapts too often is likely to experience substantial wear-and-tear of the disks[29]. We also assume that the system has a persistent memory of 1 minute or that the shift parameter is 4 minutes, which means that it retains roughly one-fifth of the observations from the previous window. We found this to work well in practise to introduce a sufficient hysteresis in the system to prevent the system from oscillations. The run-length parameters pertain to the simulation and we wanted to attain results with 99% confidence intervals upto 2 digits of accuracy. We run the simulation for a maximum of 100000 seconds of simulation time, which corresponds to little over a day. We found that most configurations converge within this time. Note that we terminate the simulation if the required confidence level is not reached in this time. We also run it for a minimum time of 10000 seconds to allow transients to die down.

5.3 Applications

In order to evaluate the system, we use both the synthetic traces generated by the b-model algorithm (described in section 4.3.1) as well as real application traces. We use I/O traces collected from two parallel applications, ESCAT [52] and sPPM [53] running on a 16-node cluster running Linux and IBM's GPFS file system [54]. These traces were obtained using the Pablo I/O trace libraries [55] which are an extension of the Pablo Performance Capture Facility (PCF) [56], developed by the Pablo group at the University of Illinois at Urbana-Champaign. They are in the Pablo SDDF format [57] and can be manipulated using the Pablo I/O analysis tools [55].

5.3.1 ESCAT

The study of low-energy electron-molecule collisions is of interest in many contexts, including aerospace applications, atmospheric studies, and the processing of materials using low-temperature plasmas (e.g., semiconductor fabrication). The Schwinger multichannel (SMC) method is an adaptation of Schwinger's variational principle for the scattering amplitude that makes it suitable for calculating low-energy electron-molecule collisions [52]. ESCAT is a parallel implementation of the Schwinger Multichannel method written in C, FORTRAN, and assembly language.

A production run is typically done in two execution phases. First, a compulsory read loads the problem definition and some initial matrices. All nodes participate in the calculation and storage of the requisite quadrature data set, with each node processing a different set of integrals. This phase is compute-intensive and is composed of a series of compute/write cycles with the write steps synchronized among the nodes. Memory limitations and the desire to checkpoint the quadrature data set for reuse in later executions prompt the writes during this phase. The second phase involves calculations that depend on the collision energy. In it, energy-dependent data structures are generated and combined with the reloaded quadrature data set to form the system of linear equations. Finally, the linear system matrices are written to disk for later solution on another machine. A detailed characterization of the ESCAT I/O pattern can be found in [9].

5.3.2 SPPM

sPPM [53] solves a 3-D gas dynamics problem on a uniform Cartesian mesh, using a simplified version of the Piecewise Parabolic Method. It is also one of the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) benchmarks. The algorithm makes use of a split scheme of X, Y, and Z Lagrangian and remap steps, which are computed as three separate sweeps through the mesh per time-step. Mes-

sage passing provides updates to ghost cells from neighboring domains three times per time-step. The code is written in Fortran-77 with some C-routines and is mostly compute-intensive.

As far as I/O is concerned, all the reads occurs in the first few seconds of the execution to read initialization data. All processors are involved in reading the initialization data, which increases the stress on the I/O system. A detailed analysis of the computational and communication characteristics of the code can be found in [58].

5.4 Results

In this section, we report results for two parallel applications: sPPM and ESCAT. We also report results for the synthetic traces generated from the b-model. In most of the graphs in this section, we look at the response time-power product. The power is calculated by computing the total energy expended by the system and dividing it by the total time for which the simulation is run. This allows us to study the energy consumed by the system irrespective of how long the system must be run for convergence reasons. Of course, this is not an issue for the application traces, as they are simply run for the duration of the trace, but it is an issue for the b-model traces which are run till the system converges or till such time we are convinced that it cannot converge.

5.4.1 b-Model Results

The b-model generates continuous, but irregular traces, which means the system should adapt almost continuously to the right configuration. In this section, we use average utilization as the metric based on which the system responds. The monitor component observes the average utilization of the system and alerts the controller.

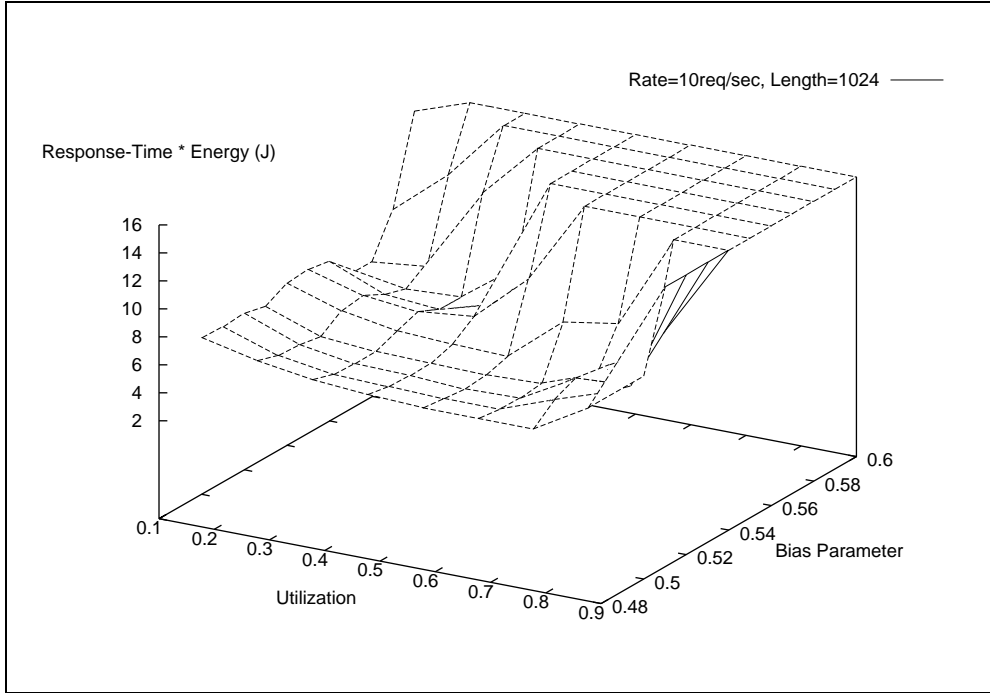


Figure 5.1: b-model Results for Rate = 10 Requests/sec, Length = 1024 blocks

If the value lies between the minimum and maximum values allowed, the controller takes no action. If it is above the maximum value, then the system is over-utilized and the controller spins up a group to decrease the average utilization. If it is below the minimum value, the controller spins down a group to save energy. For the rest of this section, we will assume that the difference between the minimum utilization value and the maximum utilization value is 0.1. This allows us to bound the response time of the system in a small interval while allowing for small variations due to bursts and transients. Henceforth, when we say utilization value, it refers to the minimum utilization value. The maximum utilization is obtained by adding 0.1 to this value.

The utilization metric has some advantages compared to the queue length or response time. For one, the utilization value is always between 0 and 1, so it is possible to set the bound on the utilization independent of the system characteristics. Also, there is a direct correspondence between the utilization value and the load on the system. The higher the utilization value, the more the load on the system and

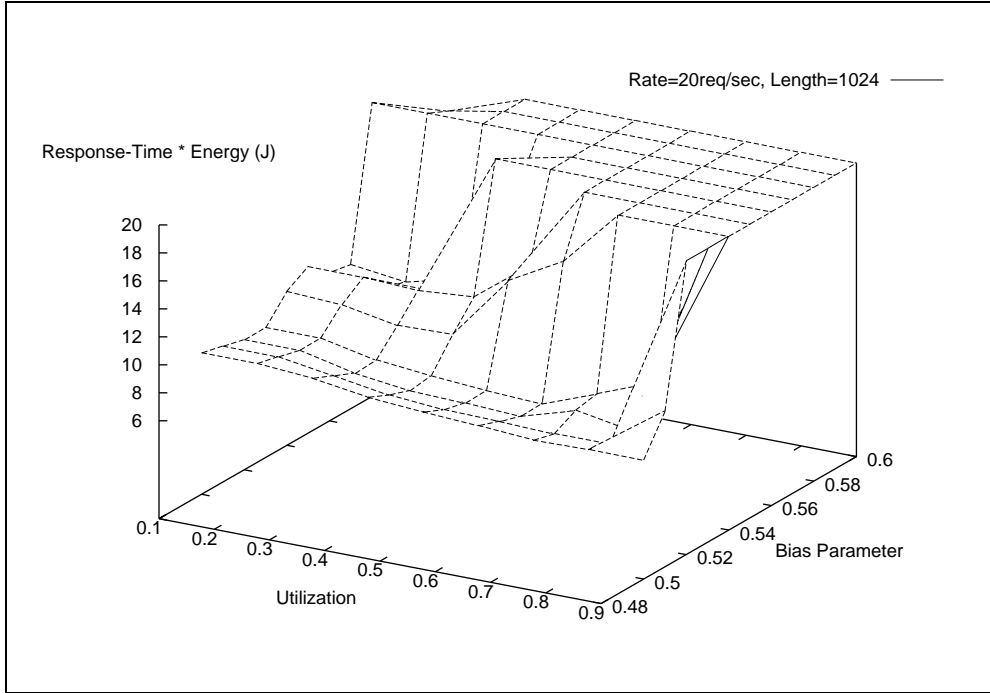


Figure 5.2: b-model Results for Rate = 20 Requests/sec, Length = 1024 blocks

the lower the utilization value, the lesser the load on the system. As the aim of power management is to adapt the system depending on the load, the choice of the utilization metric is a natural one.

The other two metrics we considered were average response time and the queue length. The response time of a request depends both on the queuing delay and the service time. The former depends on other requests and is a property of the system load, while the latter depends on the request only and is a property of its length and arrival time. It is possible that a large request may have a very high response time due to the service time component, in spite of the fact that the queue is nearly empty. This may cause the monitor to believe that the system load is high and spin up more groups, which would not help the request response time. As a result, response time is an unreliable metric and requires knowledge of both the system and the workload to choose correctly.

Similarly, the queue length of the system does not accurately represent the system

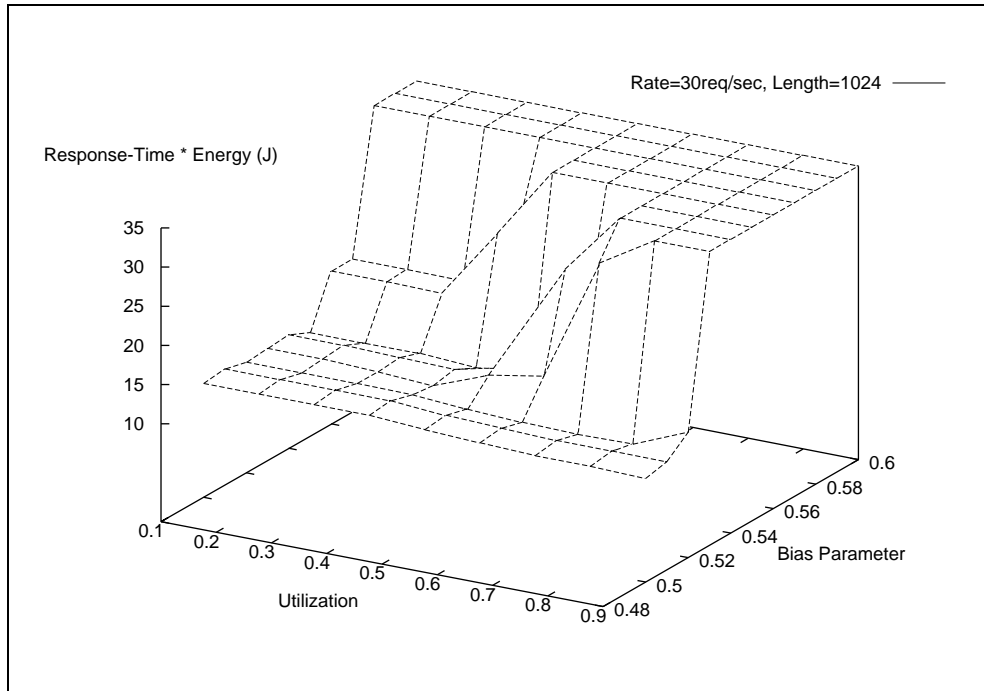


Figure 5.3: b-model Results for Rate = 30 Requests/sec, Length = 1024 blocks

load. To see this, consider a workload with a high arrival rate but consisting of very short requests. Also, assume that the system is able to deal with these requests very fast, and there is no noticeable delay for the user. But it is possible that the queue length may be large due to buffering of requests, even though the load on the system is low. Or it may be that the queue consists of a few very long requests each of which take a long time to service, and hence the load on the system is high. In both cases, queue length is not an appropriate metric based on which to adapt the system.

The disadvantage of choosing utilization as the metric for adaptation is that there is no direct correlation between utilization and response time. In the end, all the user cares about is the response-time of the system and it is up to the system administrator to choose the utilization value so that the response time of the system is acceptable to users. In this study, we assume that an average response time of less than 1 second is an acceptable delay. However, this may not always be the case, depending on the demands of the user and the length of requests. Nevertheless, we believe this is a

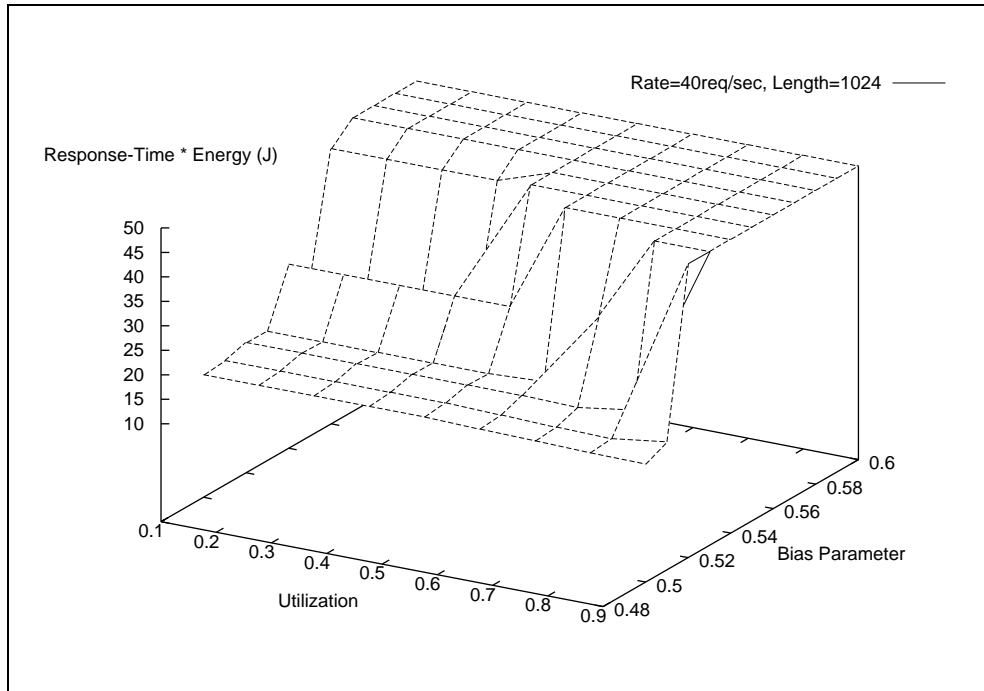


Figure 5.4: b-model Results for Rate = 40 Requests/sec, Length = 1024 blocks

reasonable assumption and that the user is unlikely to accept a system in which the response time is greater than 1 or 2 seconds.

In our study of the b-model I/O traces, we vary the bias parameter as well as the target utilization value and study the effect on the response-time power product of the system. Figures 5.1 to 5.5 show the effects of varying the utilization and the bias parameter on the response-time power product at different request rates. All these graphs assume an average request length of 1024 disk blocks. Further, we consider only traces with bias parameter between 0.5 and 0.6, as otherwise our system is unable to cope with the sudden increases and decreases in the request volume. We believe that this is a weakness of the current implementation of the adaptation mechanism, and we need to incorporate online prediction to augment the adaptation algorithm.

Let us first look at the relationship between utilization and bias for a fixed request rate. Consider figure 5.1 which plots the response-time power product of the system for a request rate of 10 requests/second. First let us consider the variation

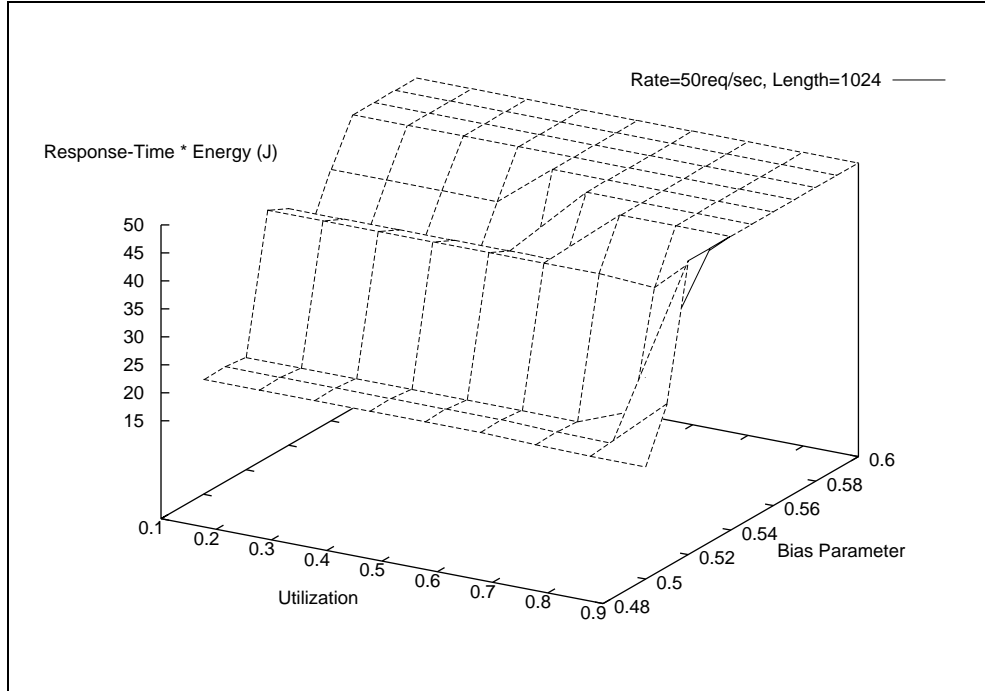


Figure 5.5: b-model Results for Rate = 50 Requests/sec, Length = 1024 blocks

of the response-time power product with the utilization value. We observe that the product first decreases with increasing the target utilization value and then starts increasing again. In other words, there is an optimum value of the utilization for a given bias parameter. For values of the utilization lesser than this value, the system is over-provisioned and wastes energy. As a result, the power term dominates and the response-time power product is more. For values of the utilization greater than this value, the system is under-provisioned and overloaded. As a result, the response-time term term dominates, and the response-time power product is more. Note that the increase in response-time power product is more to the right of the optimum value than the left, indicating that the response-time is more dominant in the response-time energy product.

We now consider what happens when we increase the bias for a given request rate. The optimum utilization value decreases with increase in the bias parameter for a given request rate. We can explain this as follows: the bias represents the

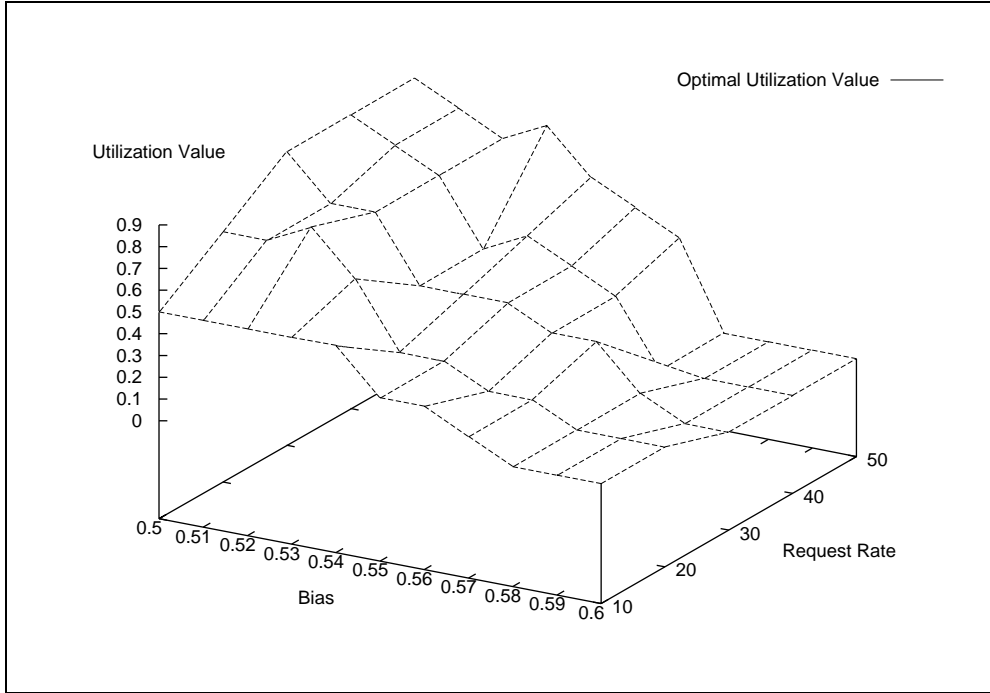


Figure 5.6: b-model Results for Optimal Utilization

burstiness of the trace, and we need to over-provision more for a more bursty trace. This is so because the more bursty the trace is, the greater the probability that there is a sudden burst of requests which overwhelm the system. The system must always have some margin of safety to deal with the sudden, unexpected bursts and cannot have just enough active replicas to deal with the average request rate of the trace. As a result, the system administrator must choose a lower target utilization for a more bursty trace than for a smooth trace, even though both have the same request rate. Another thing to observe is that the response-time power product is more for a more bursty trace at the same utilization value. This reflects the notion that increasing the variance of the input workload leads to higher response times, which in turn increases the response-time power product.

We now look at the variation of the response-time power product at different values of the request rate. In general, for a given utilization and bias parameter, the response-time power product increases with request rate. However, at higher request rates we do not observe the bell-shaped curve of the response-time power product

versus utilization for a given bias, as the curve is almost flat to the left. The reason for this is that at higher request rates, response-time dominates the response-time power product and energy becomes less important. This means that choosing the right utilization value is important at low request rates, but not so important at higher request rates, as long as it is not so large as to overload the system.

We also look at the variation of the optimum utilization value across different request rates and bias values. This is shown in figure 5.6 and is essentially derived from figures 5.1 to 5.5. We note that for a given request rate, the optimum utilization value decreases with increase in the bias value, as explained before. We also note that for a given bias value, the optimum utilization value increases with increase in request rate. Also, a value of zero for the optimum utilization implies that the system is unstable at that point, and occurs at higher values of the request rate and bias. This means that, no matter what the utilization value is, the system will not be able to handle the workload and still maintain the average response time less than one.

5.4.2 Application Results

The ESCAT and sPPM applications have both read and write requests in their I/O traces. We need to filter out the writes from the trace, as our system is designed for read-only data, and does not handle writes. However, this does not alter the characteristics of the read traffic, as the reads and writes occur in distinct phases in both applications, and are well separated from each other. Also, both applications perform synchronous I/O, and hence there is at most one outstanding request from each compute node to the I/O system. As a result, we need to make sure that the next request is issued only after the previous request is completed. We do this by calculating the computation intervals between I/O requests in the trace, and delaying the next request by this value, rather than use the arrival times from the trace directly.

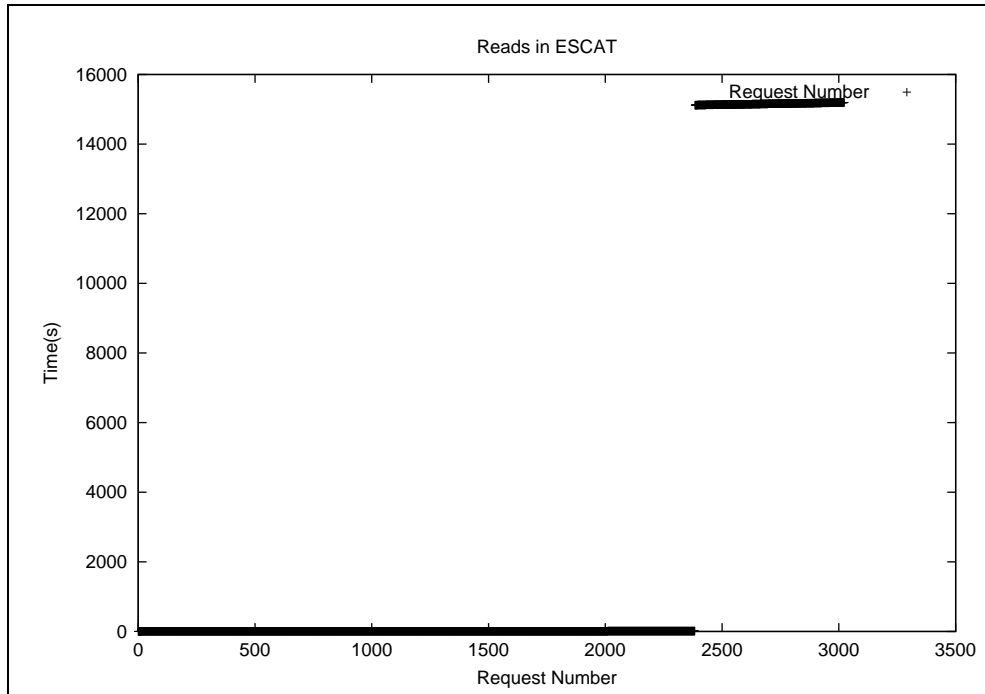


Figure 5.7: Distribution of read requests over time in ESCAT

ESCAT

In ESCAT, all the I/O is done by only one node, irrespective of how many compute nodes are there. Because the I/O is done in a synchronous manner, there is at most one outstanding request to the system, overall. In other words, even with one replica active, no request experiences a queuing delay at all. Hence, no matter how many replicas are present, it is the same as having just one replica, as there is no concurrency among requests.

Figure 5.7 shows the distribution of read requests in the ESCAT trace file with time. Most of the reads occur during the first 10 seconds at the beginning of the execution, when all the replicas are active. This can be seen in the magnified view of the first 10 seconds of the execution in figure 5.8. However, there is no decrease in the response time because of these replicas, due to the lack of concurrency. After this initial burst, there is a very long idle period during which the system adapts itself and spins down all but one replica. In this configuration, the energy consumption of the

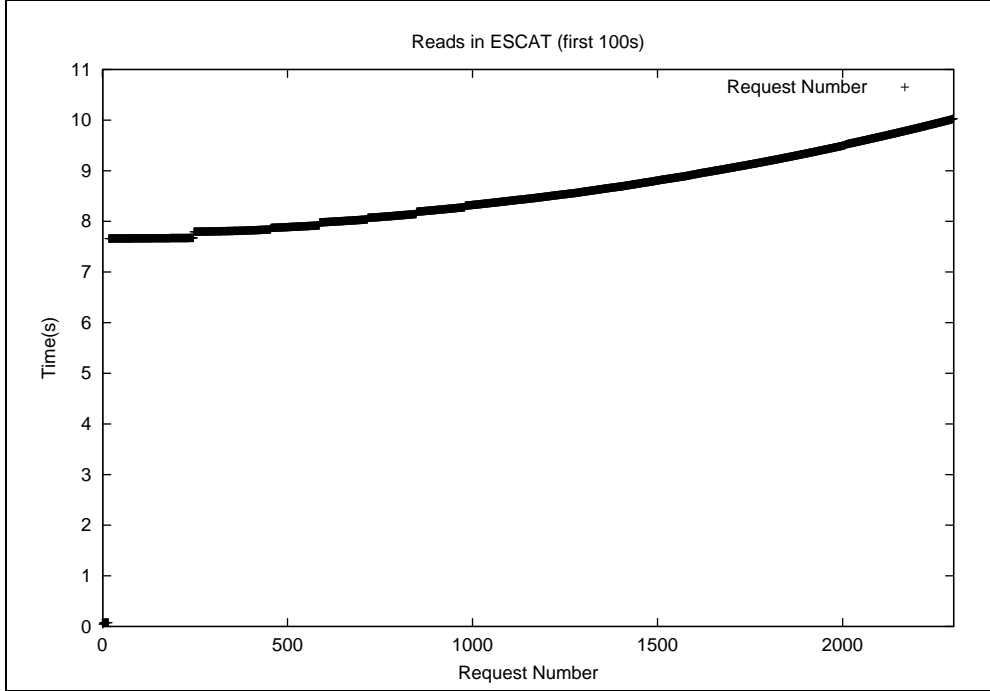


Figure 5.8: Distribution of read requests over time in ESCAT for the first 10 seconds system is minimized. Then, after about 18000 seconds at the end of the execution, there is another spurt of reads which occur. We always keep one replica active, so there is no performance degradation experienced by these reads, as they do not have to wait for replicas to spin up.

The ESCAT trace allows us to study how to trade-off performance for energy savings by choosing a proper group size. In ESCAT, there are a few very large requests which dominate the average response time, and lots of very small requests which take very little time to execute. Disk striping can decrease the execution time of these large request, by executing them in parallel. Hence, the choice of the group size dictates how much faster we can execute these large requests and reduce the overall response time. The larger the group-size, the greater is the speedup, as long as it is not too large. However, having a large group means spending more energy, as at least one group needs to be powered up all the time. So there is an optimal group size which results in the lowest response-time power product for ESCAT.

We study the response-time and the response-time power product for different

values of the group size. In these experiments, the total number of disks was kept constant and the number of replicas varied accordingly. This does not have an impact on the performance as the requests are synchronous, but impacts the initial energy consumption slightly. The greater the number of replicas present, the longer the system takes to spin them down to achieve the minimal energy configuration. Figure 5.9 shows the adaptation of the system for different values of the group size. Note that smaller group sizes indicate more replicas as the total number of disks is kept constant. We can observe that the system adapts faster for higher number of replicas (smaller group sizes) than for lower number of replicas (larger group sizes).

Figures 5.10 and 5.11 show the effect of different group sizes on the response-time and the response-time power product of the system. From the response-time curve, we can see that the average response time decreases as the group size increases. However, the energy consumption is more for larger group sizes and this can be seen in the response-time power product curve, which decreases with increasing group size up to a point, and then starts increasing. The optimal group size which yields the lowest response-time power product is between 8 to 16 disks. Also, choosing a larger group size than this optimal value is better than choosing a smaller group size, as the curve shoots up sharply to the left of the optimal value. The increase in response-time power product is more gradual with increase in group size, suggesting that response-time is more important than power in deciding the shape of the curve.

SPPM

sPPM is similar to ESCAT in that most of the reads occur during the first few seconds of execution (first 25 seconds). However, it is different in that all the nodes are involved in the I/O in a synchronous fashion, unlike in the case of ESCAT, where a single node did all the I/O. Also, there is no second round of reads at the end of the execution as was the case with ESCAT. Figure 5.12 shows the distribution of read

requests over time in the sPPM trace. Figure 5.13 shows the breakdown of requests in the trace according to which node is performing the I/O. In this figure, the X-axis represents the processor (node) number and the Y-axis represents time.

For sPPM, we varied the number of nodes involved in the I/O as well as the group size. We obtained traces of sPPM with 1, 2, 4, 8 and 16 nodes and plotted the response-time and the response-time power product of the system with these traces at different group sizes. Figures 5.15 and 5.16 shows the results for the response time and the response time-energy product and Figure 5.14 shows the adaptation of the system for different values of the group size when the number of nodes involved in the I/O is fixed at 8. As in the case of ESCAT, the system adapts faster for larger group sizes than for smaller ones.

The behavior of sPPM with a single processor is similar to that of ESCAT and the optimal group size is between 16 and 32 disks. However, this optimal size changes when the total number of nodes is varied. To understand this, we need to understand the effect of changing the number of nodes in terms of the analytical model proposed in section 3.4. By increasing the number of nodes, we are increasing the number of simultaneous, outstanding requests in the system. In other words, we are increasing the request arrival rate of the system, though not in a straightforward way (as the rate also depends on how long the requests take to complete). We saw from the analytical model that at higher request rates, a smaller stripe width is better for the throughput of the system. This corresponds to choosing a smaller group size for the system as the number of nodes increases. This is true for both the response-time graph and the response-time power product graph and can be seen from figures 5.15 and 5.16.

However, the optimal value of group size when considering response-time alone is different from the optimal value of group size when considering the response-time power product. For example, for 4 nodes, the optimal group size when considering response-time alone is between 64 and 128 disks. But when we consider the response-

time power product also, it is between 32 and 64 disks. In general, the optimal group size is lesser when we consider the response-time power product as compared to the response-time alone, for a given number of nodes. This is so because the power consumption goes up for larger group sizes, and the average response time also increases beyond a point. In other words, the response time decreases and then increases, the power steadily increases, and the product of the two also decreases and then increases but at a much smaller value than the response time alone.

5.4.3 Summary

There are a number of important conclusions we can draw from these results. First, for a given application, there is an optimal group size which depends on the number of I/O nodes in the application as well as the average request length. Also, this optimal group size is different when considering the response-time power product as opposed to considering the response-time alone. Further, when the system adapts to bursty, irregular traces it must always adapt with a margin of safety depending on the burstiness of the trace. Utilization is a stable metric based on which the system can be adapted, and choosing the right utilization value for the system is more important at low request rates than at high rates. Finally, response-time is more important than energy in deciding the response-time power product and it is better to dynamically over-provision the system than to under-provision it.

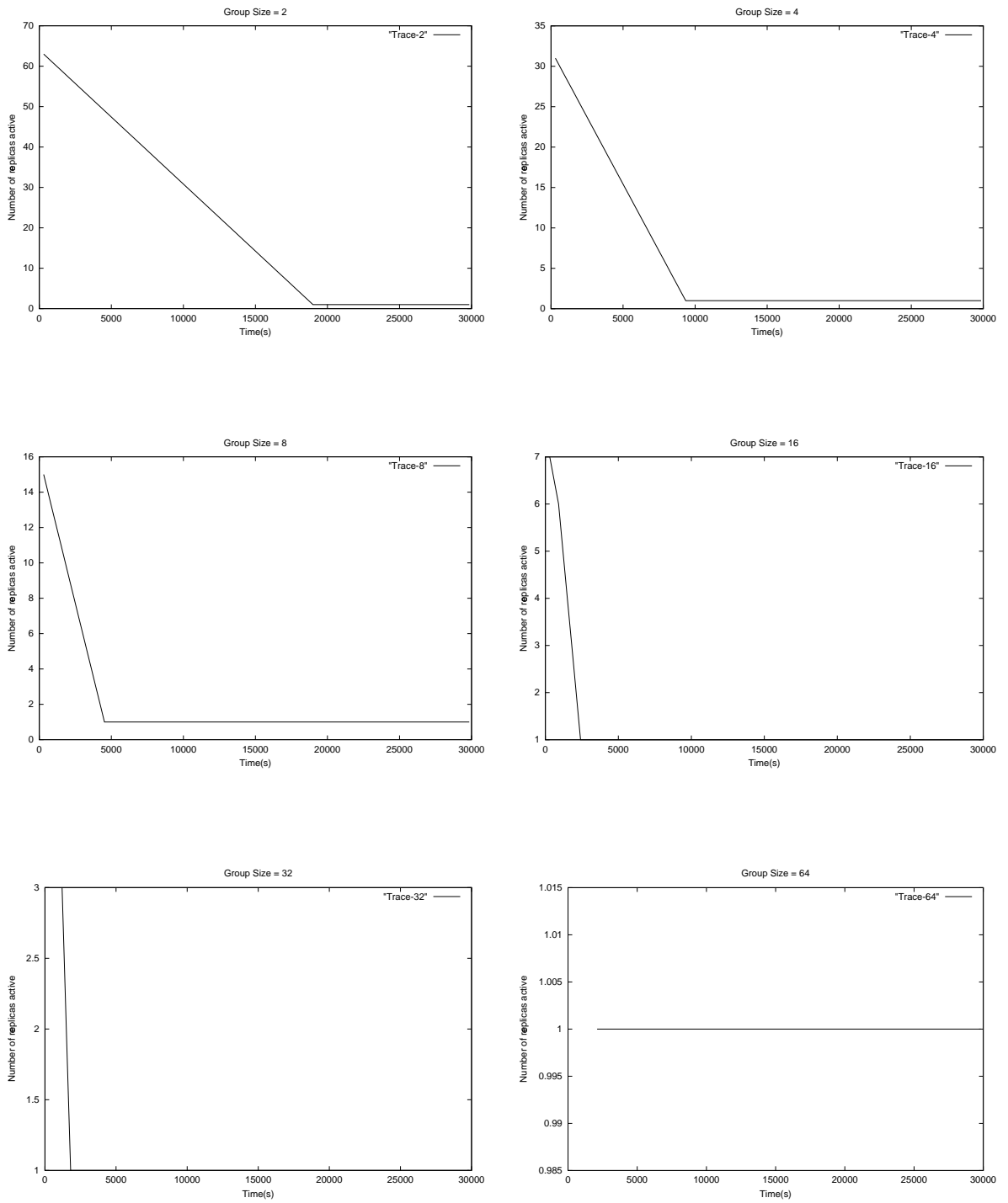


Figure 5.9: Figure showing the adaptation of the system to the escat trace for different group sizes

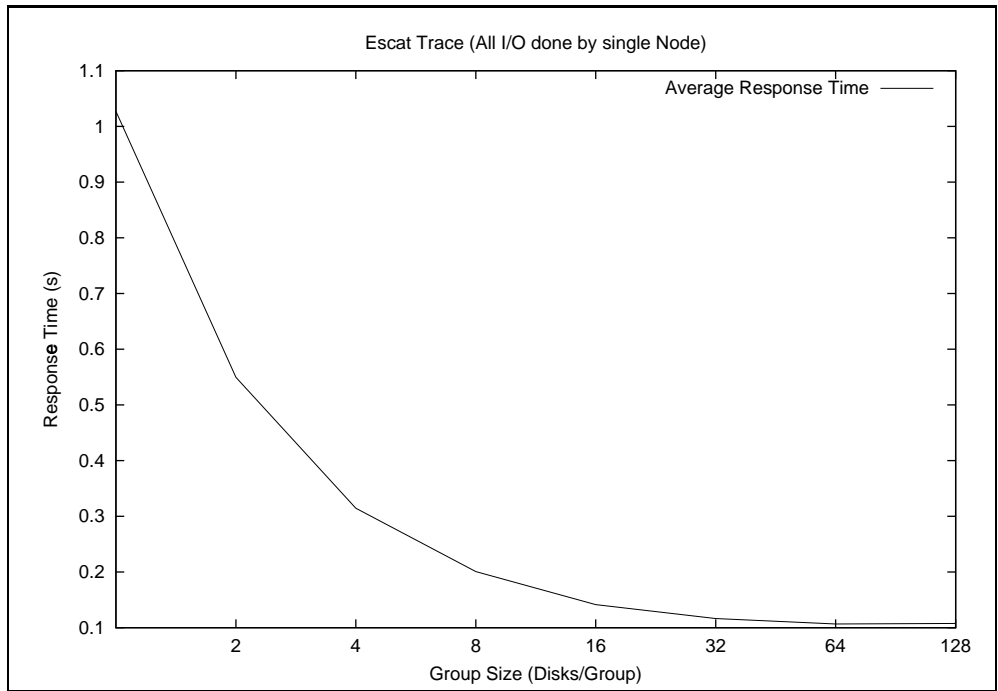


Figure 5.10: ESCAT Results for Response-Time

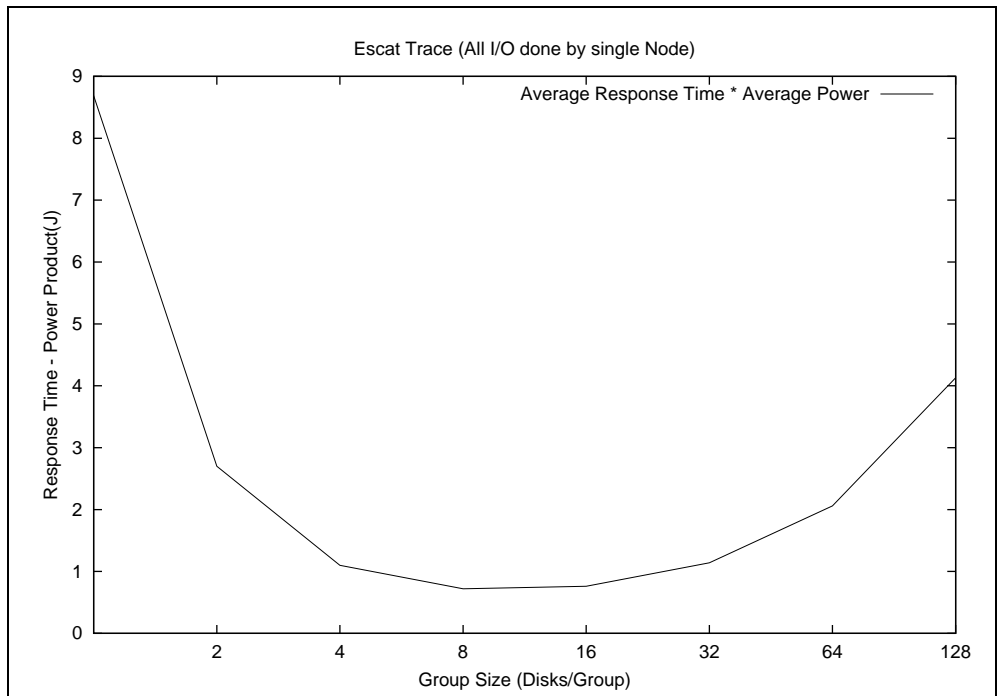


Figure 5.11: ESCAT Results for the Response-Time Power Product

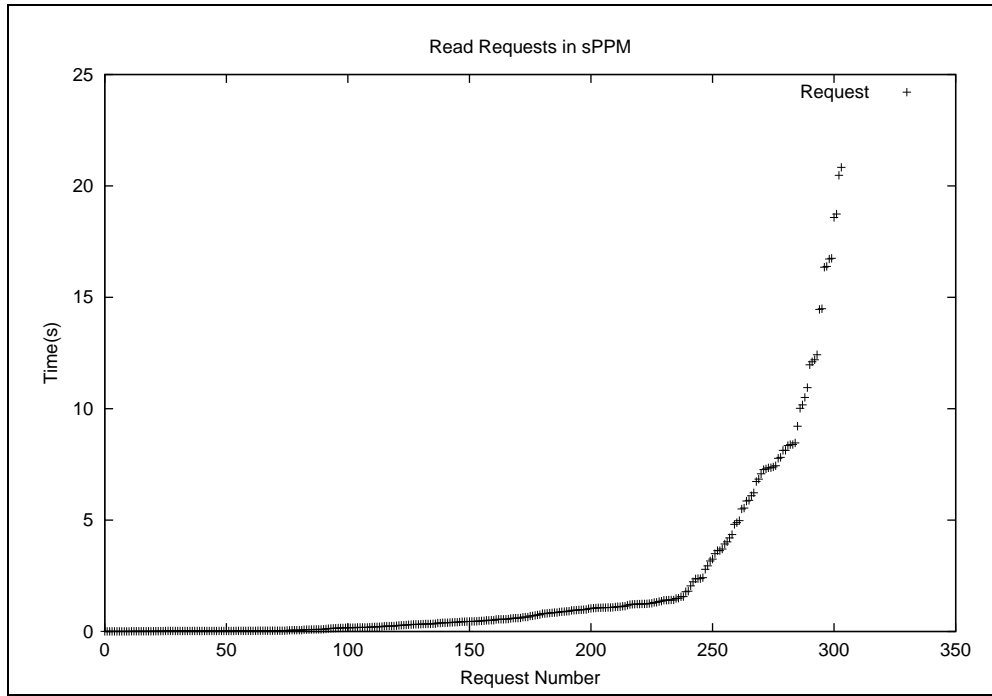


Figure 5.12: Distribution of read requests over time in sPPM

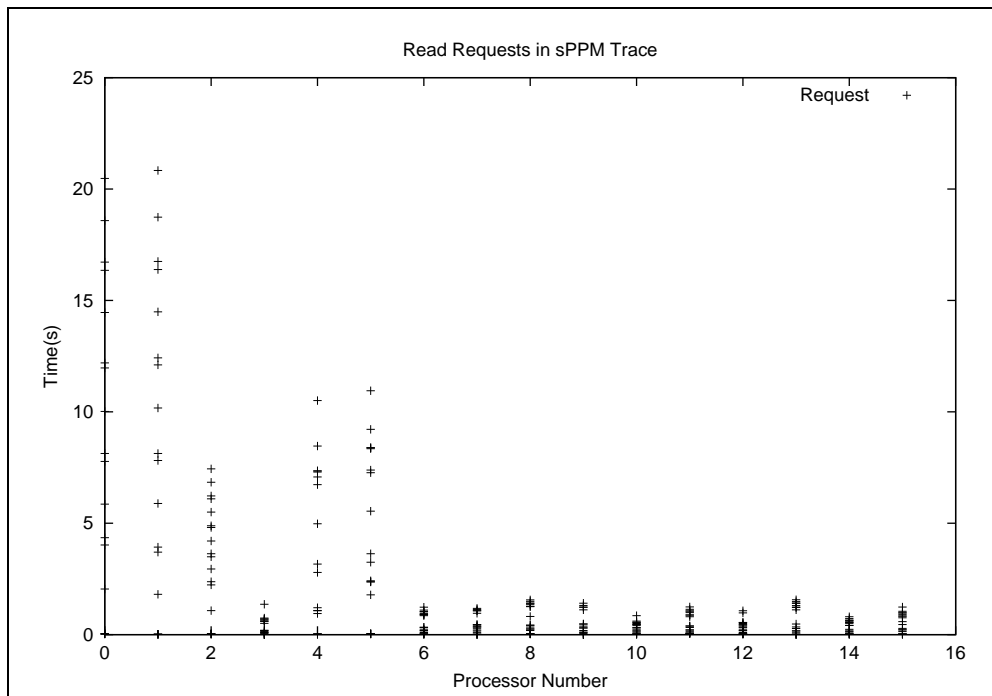


Figure 5.13: Breakdown of read requests in sPPM for each processor

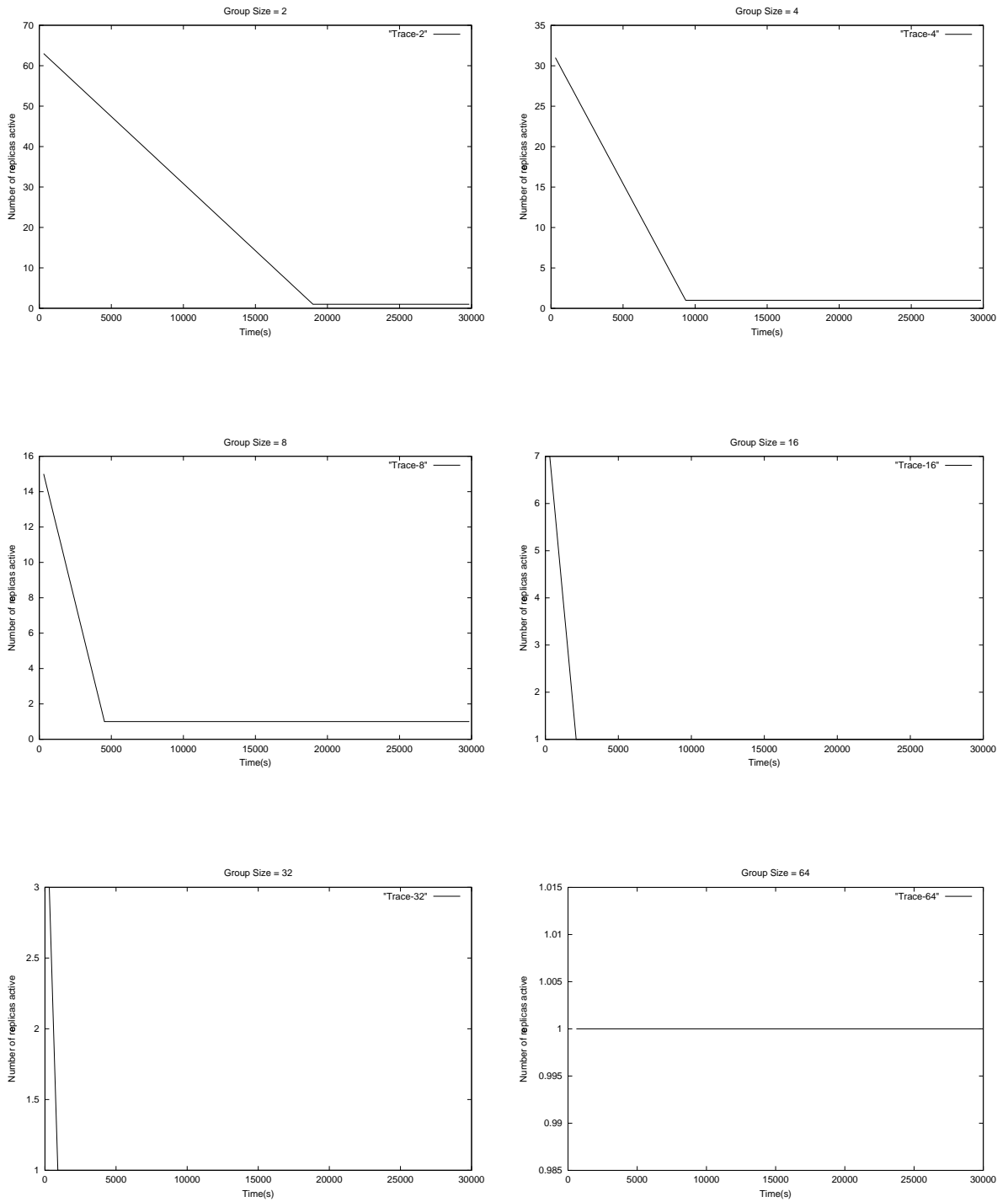


Figure 5.14: Figure showing the adaptation of the system to the spm trace for different group sizes. The number of nodes involved in I/O is fixed at 8

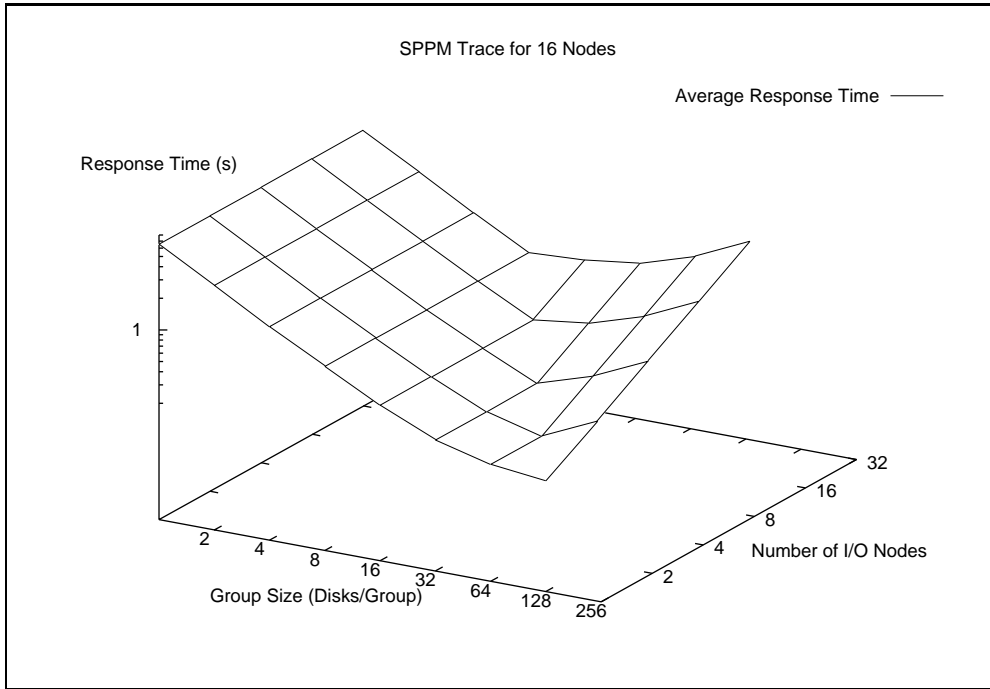


Figure 5.15: SPPM Results for Response-Time

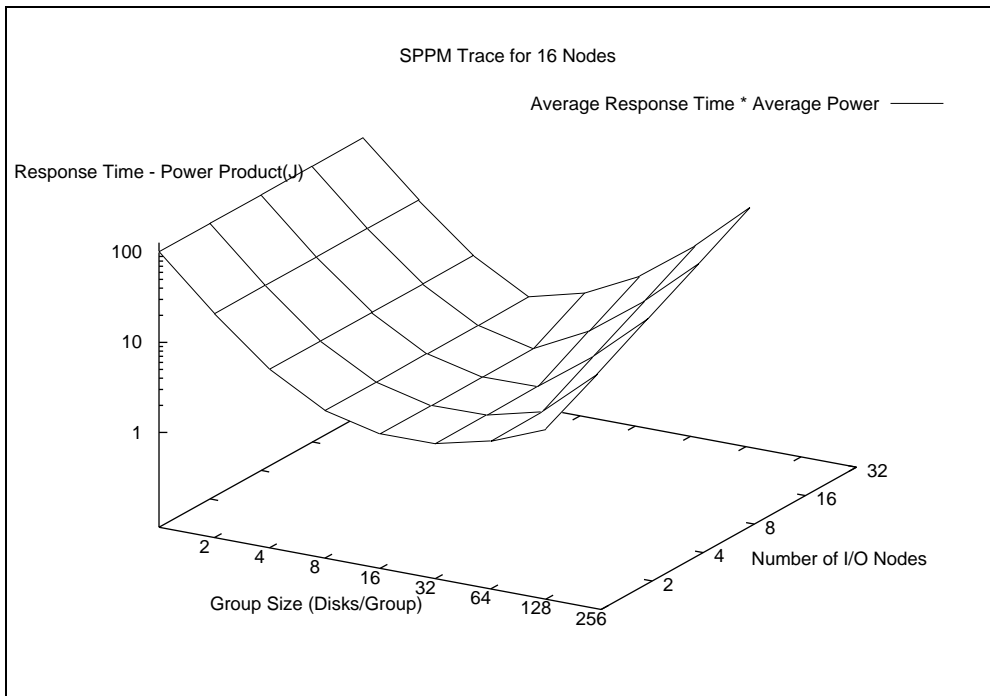


Figure 5.16: SPPM Results for the Response-Time Power Product

Chapter 6

Conclusions and Extensions

6.1 Overview

This chapter presents our conclusions and possible directions for extension of this thesis.

6.2 Conclusions

In this thesis, we studied the design and implementation of a power-aware parallel I/O system. We derived an analytical model for the system and analyzed the trade-offs in its design. We then implemented a simulation model of the system, which was used to validate the analytical model and consider scenarios which the analytical model did not capture. In particular, we experimented with real-world parallel applications and irregular and bursty synthetic traces. We found that by carefully tuning the system parameters and configuring it for a particular class of workloads, we could achieve substantial energy savings with little or no performance loss. We also found that the amount of energy savings obtained is heavily dependent on the workload and regular workloads offer better power savings than bursty workloads. However, parallel I/O workloads are often bursty and irregular and systems must be designed

with overprovisioning to handle them effectively.

6.3 Extensions

One direction of extension of this system is to automate the choice of the group size, depending on the workload characteristics. Currently, the designer has to experiment with various parameters and configure the system for a particular type of workload. We could automate the system configuration based on a rule-base derived from the analytical model. As this is a stochastic system, we could use fuzzy-logic to disambiguate between rules. In fact, we could redundantly stripe the data across groups of different size, and the appropriate group size could be picked dynamically at runtime. This is similar to the redundant striping scheme proposed by Simitci and Reed [18].

The current adaptation mechanism is very simple and is unable to cope with very bursty workloads as was illustrated in section 5.4.1. It needs to be augmented to respond more quickly to sudden bursts, and at the same time, not be subject to fluctuations in the presence of noise. The use of some formal control theory might help here, but even that would involve considerable tuning to get it to work right. One solution may be the use of sophisticated prediction techniques to predict sudden bursts in I/O activity. Time-series analysis is one technique which has been shown to be effective in predicting I/O bursts for parallel applications [24].

An orthogonal approach to reducing the burstiness of the workload is to employ aggressive prefetching and caching. In our system, we assumed that there is no prefetching or caching done by the application or the system. A real parallel file system, however, would prefetch data items in the cache for performance reasons. However, prefetching for power-savings is not the same from prefetching for performance enhancement, and the tradeoffs are different, as shown in Zhu et. al.

[40]. The problem is that we want a certain amount of burstiness for saving energy, but not so much that the system has to be dynamically over-provisioned to meet the performance target. We would like to have predictable burstiness, so that the groups could be spun up in advance in anticipation of the bursts. A good prefetching scheme should issue requests to the system in bursts of just the right size so as not to overwhelm the system, and alert the system well in advance, to adapt to the burst.

In our system, we make the assumption that all traffic is read-only. Even though this is predominantly the case for data sets used with scientific codes, we would also like to support writes as well. This would be especially useful for checkpoint data written by these codes, which are read in case of failures. In order to support writes, we would need a coherence mechanism to keep the data consistent across replicas. One way to do this is to buffer writes for spun-down groups, and synchronize them when the group is spun-up again. This would increase the cost of spinning up a group as it would now need to be offline till it is synchronized and brought up to date. The delay experienced by this process would depend on the volume of writes in the intervening period the group was spun down. This is similar to the approach taken by Carrera et. al [36].

We could also improve the simulation model of the system to make it more accurate. Instead of modeling the disk analytically, we could simulate the actual operation of the disk using a validated simulator like DiskSim [59]. This would allow us to model both the disk and the interconnect in great detail but would have a large memory and time cost, as we would likely need to simulate hundreds or thousands of disks in our system. We could also model the energy consumed more accurately using Dempsey [60], a tool which extends DiskSim to model energy as well as performance. It currently does this for laptop disks, but it wouldn't be too difficult to extend it for server disks as well.

The most important step would be to implement this system on a real parallel file

system like PVFS [5], instead of just simulating it. However, this would involve building a cluster with disks that could be spun-down on demand, and current technology for server disks does not permit this. However, we could simulate disk spin-down in a real system by not sending requests to spun-down groups. With this approach, we would need to keep track of the energy consumed by simulation, but we could use the real implementation for calculation of response times.

References

- [1] T Anderson, D Culler, and D Patterson, “A case for networks of workstations,” in *IEEE Micro*, February 1995.
- [2] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer, “BEOWULF: A parallel workstation for scientific computation,” in *Proceedings of the 24th International Conference on Parallel Processing*, Oconomowoc, WI, 1995, pp. I:11–14.
- [3] Message Passing Interface Forum, “MPI: A message-passing interface standard,” *International Journal of Supercomputer Applications*, vol. 8(3/4), 1994.
- [4] V. S. Sunderam, “PVM: a framework for parallel distributed computing,” *Concurrency, Practice and Experience*, vol. 2, no. 4, pp. 315–340, 1990.
- [5] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur, “PVFS: A parallel file system for linux clusters,” in *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, 2000, pp. 317–327, USENIX Association.
- [6] G. Bell and J. Gray, “What’s next in high-performance computing?,” *Communications of the ACM*, vol. 45, no. 2, pp. 91–95, February 2002.
- [7] E. Weigle W. Feng, M. Warren, “The Bladed Beowulf: A cost-effective alternative to traditional Beowulfs,” in *Proceedings of IEEE Cluster 2002*, September 2002.
- [8] David Kotz and Ravi Jain, “I/O in parallel and distributed systems,” in *Encyclopedia of Computer Science and Technology*, Allen Kent and James G. Williams, Eds., vol. 40, pp. 141–154. Marcel Dekker, Inc., 1999, Supplement 25.
- [9] Phyllis E. Crandall, Ruth A. Aydt, Andrew A. Chien, and Daniel A. Reed, “Input/output characteristics of scalable parallel applications,” in *Proceedings of Supercomputing '95*, San Diego, CA, 1995, IEEE Computer Society Press.
- [10] E. Smirni and D. A. Reed, “Workload characterization of input/output intensive parallel applications,” in *Proceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. 1997, vol. 1245, pp. 169–180, Springer-Verlag.

- [11] Rajesh Bordawekar, Steven Landherr, Don Capps, and Mark Davis, “Experimental evaluation of the Hewlett-Packard Exemplar file system,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, no. 3, pp. 21–28, December 1997.
- [12] P. Corbett, D. Feitelson, J.-P. Prost, G. Almasi, S. J. Baylor, A. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. Herr, J. Kavaky, T. Morgan, and A. Zlotek, “Parallel file systems for the IBM SP computers,” *IBM Systems Journal*, vol. 34, no. 2, pp. 222–248, January 1995.
- [13] Peter F. Corbett and Dror G. Feitelson, “The Vesta parallel file system,” in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, Hai Jin, Toni Cortes, and Rajkumar Buyya, Eds., pp. 285–308. IEEE Computer Society Press and Wiley, New York, NY, 2001.
- [14] Intel Scalable Systems Division, “Paragon system user’s guide,” Order Number 312489-004, May 1995.
- [15] Susan J. LoVerso, Marshall Isman, Andy Nanopoulos, William Nesheim, Ewan D. Milne, and Richard Wheeler, “SFS: A parallel file system for the CM-5,” in *Proceedings of the 1993 Summer USENIX Technical Conference*, 1993, pp. 291–305.
- [16] Steven A. Moyer and V. S. Sunderam., “PIOUS: A scalable parallel I/O system for distributed computing environments,” in *Proceedings of the Scalable High-Performance Computing Conference*, 1994, pp. 71–78.
- [17] Nils Nieuwejaar and David Kotz, “The Galley parallel file system,” in *Proceedings of the 10th ACM International Conference on Supercomputing (ICS)*, Philadelphia, PA, 1996, pp. 374–381, ACM Press.
- [18] Huseyin Simitci and Daniel A. Reed, “Adaptive disk striping for parallel input/output,” in *Proceedings of the Seventh NASA Goddard Conference on Mass Storage Systems*, San Diego, CA, 1999, pp. 88–102, IEEE Computer Society Press.
- [19] K. Salem and H. Garcia-Molina, “Disk striping,” in *Proceedings of Data Engineering ’86*, 1986, pp. 336–342.
- [20] P. H. Smith and J. Van Rosendale, “Data and visualization corridors,” Tech. Rep. CACR-164, Caltech, September 1998.
- [21] T. Sterling, P. Messina, and P. H. Smith, *Enabling Technologies for Petaflops Computing*, The MIT Press, Cambridge, MA, 1995.
- [22] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony, “The case for power management in web servers,” in *In Power-Aware Computing*, Robert Graybill and Rami Melhem, Eds., Series in Computer Science. Kluwer/Plenum, January 2002.

- [23] P. Bohrer, D. Cohn, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, R. Rajamony, F. Rawson, and E. Van Hensbergen, “Energy conservation for servers,” in *Proceedings of the IEEE Workshop on Power Management for Real-Time and Embedded Systems*, May 2001.
- [24] Nancy Ngoc Tran, *Automatic Arima Time Series Modeling And Forecasting Adaptive Input/Output Prefetching*, Ph.D. thesis, University of Illinois at Urbana-Champaign, December 2001.
- [25] SGI white paper, “XFS: A next generation journaled 64-bit filesystem with guaranteed rate I/O,” .
- [26] James V. Huber, Jr., Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal, “PPFS: A high performance portable parallel file system,” in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, Hai Jin, Toni Cortes, and Rajkumar Buyya, Eds., pp. 330–343. IEEE Computer Society Press and Wiley, New York, NY, 2001.
- [27] Huseyin Simitci, Daniel A. Reed, Ryan Fox, Mario Medina, James Oly, Nancy Tran, and Guoyi Wang, “A framework for adaptive storage input/output on computational grids,” in *Proceedings of the 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP)*, April 1999.
- [28] Jacob R. Lorch and Alan Jay Smith, “Software strategies for portable computer energy management,” *IEEE Personal Communications Magazine*, vol. 5, no. 3, pp. 60–73, June 1998.
- [29] Paul M. Greenawalt, “Modeling power management for hard disks,” in *Proceedings of the Second Int’l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, February 1994, pp. 62–66.
- [30] Kester Li, Roger Kumpf, Paul Horton, and Thomas E. Anderson, “A quantitative analysis of disk drive power management in portable computers,” in *USENIX Winter*, 1994, pp. 279–291.
- [31] Fred Douglass, P. Krishnan, and Brian Marsh, “Thwarting the power-hungry disk,” in *USENIX Winter*, 1994, pp. 292–306.
- [32] Fred Douglass, Padmanabhan Krishnan, and Brian Bershad, “Adaptive disk spin-down policies for mobile computers,” in *Proceedings of 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.
- [33] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod, “A dynamic disk spin-down technique for mobile computing,” in *Mobile Computing and Networking*, 1996, pp. 130–142.

- [34] Andreas Weissel, Bjorn Beutel, and Frank Bellosa, “Cooperative I/O: A novel I/O semantics for energy-aware applications,” in *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI’02)*, December 2002.
- [35] A. Papathanasiou and M. Scott, “Increasing disk burstiness for energy efficiency,” Tech. Rep. 792, Department of Computer Science, University of Rochester, November 2002.
- [36] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini, “Conserving disk energy in network servers,” in *Proceedings of the 17th ACM International Conference on Supercomputing (ICS)*, June 2003.
- [37] D. Collarelli, , and D. Grunwald, “Massive arrays of idle disks for storage archives,” in *Proceedings of SC’2002*, November 2002.
- [38] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M. J. Irwin, “Interplay of energy and performance for disk arrays running transaction processing workloads,” in *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, March 2003.
- [39] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut T. Kandemir, and Hubertus Franke, “DRPM: Dynamic speed control for power management in server class disks,” in *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*, 2003, pp. 169–179.
- [40] Qingbo Zhu, Francis M. David, Christo Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao, “Reducing energy consumption of disk storage using power-aware cache management,” in *Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA-10)*, February 2004.
- [41] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, “Load balancing and unbalancing for power and performance in cluster-based systems,” Tech. Rep. DCS-TR-440, Department of Computer Science, Rutgers University, 2001.
- [42] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle, “Managing energy and server resources in hosting centres,” in *Symposium on Operating Systems Principles*, 2001, pp. 103–116.
- [43] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony, “Energy-efficient server clusters,” in *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.
- [44] M. Elnozahy, M. Kistler, and R. Rajamony, “Energy conservation policies for web servers,” in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.
- [45] W. Feng M. Warren, E. Weigle, “High-density computing: A 240-node Beowulf in one cubic meter,” in *Proceedings of SC2002*, November 2002.

- [46] A. Klaiber, “The technology behind Crusoe processors,” Transmeta, White Paper, January 2000.
- [47] Mengzhi Wang, Tara M. Madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutsos, “Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic,” in *Proceedings of the 18th International Conference on Data Engineering(ICDE)*, 2002.
- [48] Maria E. Gomez and Vicente Santoja, “Analysis of self-similarity in I/O workload using structural modeling,” in *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication System (MASCOTS)*, 1999, pp. 234–242.
- [49] Mesquite Software Inc., “CSIM18 simulation engine (c++ version),” .
- [50] Mesquite Software Inc., “C++/CSim user’s guide,” 1994.
- [51] A.M. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw Hill, 1991.
- [52] WINSTEAD C. and V. McCoY, “Studies of electron-molecule collisions on massively parallel computers,” in *Modern Electronic Structure Theory*, D. R. Yarkony, Ed., vol. 2. World Scientific, 1994.
- [53] A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, M. A. Duchauneau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White, “Very high resolution simulation of compressible turbulence on the IBM-SP system,” in *Proceedings of SC’99*, 1999.
- [54] Frank Schmuck and Roger Haskin, “GPFS: A shared-disk file system for large computing clusters,” in *Proceedings of the First Conference on File and Storage Technologies (FAST)*, January 2002, pp. 231–244.
- [55] R. A. Aydt, “A user’s guide to Pablo I/O instrumentation,” Tech. Rep., University of Illinois at Urbana-Champaign, Department of Computer Science, December 1994.
- [56] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera, “Scalable Performance Analysis: The Pablo Performance Analysis Environment,” in *Proceedings of the Scalable Parallel Libraries Conference*. 1993, pp. 104–113, IEEE Computer Society.
- [57] R. Aydt, “The Pablo Self-Defining Data Format,” Tech. Rep., University of Illinois at Urbana-Champaign, Department of Computer Science, February 1993.
- [58] Jeffrey Vetter, “Performance analysis of distributed applications using automatic classification of communication inefficiencies,” in *International Conference on Supercomputing (ICS)*, 2000, pp. 245–254.

- [59] S. Bucy and G. R. Ganger, “The Disksim simulation environment version 3.0 reference manual,” Tech. Rep. CMU–CS–03–102, Department of Computer Science Carnegie-Mellon University, Pittsburgh, PA, January 2003.
- [60] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang, “Modeling hard-disk power consumption,” in *Proceedings of the Second Conference on File and Storage Technologies*, March 2003.