Universidad Carlos III de Madrid

Tesis doctoral

New techniques to model energy-aware I/O architectures based on SSD and hard disk drives

Autor: Laura Prada Camacho
Directores: Jesús Carretero Pérez
José Daniel García Sánchez

Departamento de Informática

Leganés, 2012

Universidad Carlos III de Madrid

Ph.D. Thesis

New techniques to model energy-aware I/O architectures based on SSD and hard disk drives

Author: Laura Prada Camacho
Advisors: Jesús Carretero Pérez
José Daniel García Sánchez

Departamento de Informática

Leganés, 2012

TESIS DOCTORAL

New techniques to model energy-aware I/O architectures based on SSD and hard disk drives

AUTOR: Laura Prada Camacho
DIRECTORES: Jesús Carretero Pérez
José Daniel García Sánchez

TRIBUNAL CALIFICADOR

PRESIDENTE:

VOCAL:

VOCAL:

VOCAL:

SECRETARIO:

CALIFICACION:

Leganés, a de de 2012

To my grandparents, ***Francisco*** *and* ***Crisanta****, for their everlasting support.*

Acknowledgements

Many people have contributed to make this Ph.D. process easier.

I would like to thank my advisors, Jesús Carretero and José Daniel García, for giving me the opportunity of writing this thesis, and allowing me to be a member of their research group.

Other members of ARCOS research group at University Carlos III of Madrid have contributed to make all this time easier, including Alejandro Calderón, Javier García, Javier Fernández, Félix García, Luis Miguel Sánchez, and Alejandra Rodríguez. Most of them were encouraging, helpful, and inspired me to improve myself.

I will always be grateful to Dr. Reddy for allowing me to participate in his research projects, and making me feel as one more of his students. During six months at Texas A&M, his guidance helped me to understand many ideas, and showed me a totally different point of view of what research is. I will also be grateful to Carolyn Warzon (Dr. Reddy's secretary) for being so helpful, and becoming a good friend and confidante.

I thank my parents and grandparents, for being a model of hard working and humility. I thank my sister, María Ángeles, for believing in me, unconditionally. I am grateful to my dearest friends, Laura Río, Leticia, and Javito, for being there, supporting me, from the beginning.

I am grateful to my american family, for making me feel as a part of it. Bobbie and Reagan McDonald, and Naomi Jones, were and will always be my american parents. I thank my neighbour (in Bryan) and friend, Adrian Garza, for being so helpful, and always including me in his great plans. I am also grateful to Sandeep Yadav, for his great advices, encouraging words, and providing feedback in my first journal paper. I thank my best office mate ever, Boyuan Yan, for always making me feel comfortable.

Finally, I would like to thank all the people who, somehow or other, have contributed to the completion of this thesis.

Laura Prada Camacho.

Abstract

For years, performance improvements at the computer I/O subsystem and at other subsystems have advanced at their own pace, being less the improvements at the I/O subsystem, and making the overall system speed dependant of the I/O subsystem speed.

One of the main factors for this imbalance is the inherent nature of disk drives, which has allowed big advances in disk densities, but not so many in disk performance. Thus, to improve I/O subsystem performance, disk drives have become a goal of study for many researchers, having to use, in some cases, different kind of models. Other research studies aim to improve I/O subsystem performance by tuning more abstract I/O levels. Since disk drives lay behind those levels, real disk drives or just models need to be used.

One of the most common techniques to evaluate the performance of a computer I/O subsystem is found on detailed simulation models including specific features of storage devices like disk geometry, zone splitting, caching, read-ahead buffers and request reordering. However, as soon as a new technological innovation is added, those models need to be reworked to include new characteristics, making difficult to have general models up to date.

Our alternative is modeling a storage device as a black-box probabilistic model, where the storage device itself, its interface and the interconnection mechanisms are modeled as a single stochastic process, defining the service time as a random variable with an unknown distribution. This approach allows generating disk service times needing less computational power by means of a variate generator included in a simulator. This approach allows to reach a greater scalability in I/O subsystems performance evaluations by means of simulation.

Lately, energy saving for computing systems has become an important need. In mobile computers, the battery life is limited to a certain amount of time, and not wasting energy at certain parts would extend the usage of the computer. Here, again the computer I/O subsystem has pointed out as field of study, because disk drives, which are a main part of it, are one of the most power consuming elements due to their mechanical nature. In server or enterprise computers, where the number of disks increase considerably, power saving may reduce cooling requirements for heat dissipation and thus, great monetary costs.

This dissertation also considers the question of saving energy in the disk drive, by making advantage of diverse devices in hybrid storage systems, composed of Solid State Disks (SSDs) and Disk drives. SSDs and Disk drives offer different power characteristics, being SSDs much less power consuming than disk drives. In this thesis, several techniques that use SSDs as supporting devices for Disk drives, are proposed. Various options for managing SSDs and Disk devices in such hybrid systems are examinated, and it is shown that the proposed methods save energy and monetary costs in diverse scenarios. A simulator composed of Disks and SSD devices was implemented. This thesis studies the design and evaluation of the proposed approaches with the help of realistic workloads.

Resumen

Durante años, las mejoras de rendimiento en el subsystema de E/S del ordenador y en otros subsistemas han avanzado a su propio ritmo, siendo menores las mejoras en el subsistema de E/S, y provocando que la velocidad global del sistema dependa de la velocidad del subsistema de E/S.

Uno de los factores principales de este desequilibrio es la naturaleza inherente de las unidades de disco, la cual que ha permitido grandes avances en las densidades de disco, pero no así en su rendimiento. Por lo tanto, para mejorar el rendimiento del subsistema de E/S, las unidades de disco se han convertido en objetivo de estudio para muchos investigadores, que se ven obligados a utilizar, en algunos casos, diferentes tipos de modelos o simuladores. Otros estudios de investigación tienen como objetivo mejorar el rendimiento del subsistema de E/S, estudiando otros niveles más abstractos. Como los dispositivos de disco siguen estando detrás de esos niveles, tanto discos reales como modelos pueden usarse para esos estudios.

Una de las técnicas más comunes para evaluar el rendimiento del subsistema de E/S de un ordenador se ha encontrado en los modelos de simulación detallada, los cuales modelan características específicas de los dispositivos de almacenamiento como la geometría del disco, la división en zonas, el almacenamiento en caché, el comportamiento de los buffers de lectura anticipada y la reordenación de solicitudes. Sin embargo, cuando se agregan innovaciones tecnológicas, los modelos tienen que ser revisados a fin de incluir nuevas características que incorporen dichas innovaciones, y ésto hace difícil el tener modelos generales actualizados.

Nuestra alternativa es el modelado de un dispositivo de almacenamiento como un modelo probabilístico de caja negra, donde el dispositivo de almacenamiento en sí, su interfaz y sus mecanismos de interconexión se tratan como un proceso estocástico, definiendo el tiempo de servicio como una variable aleatoria con una distribución desconocida. Este enfoque permite la generación de los tiempos de servicio del disco, de forma que se necesite menos potencia de cálculo a través del uso de un generador de variable aleatoria incluido en un simulador. De este modo, se permite alcanzar una mayor escalabilidad en la evaluación del rendimiento del subsistema de E/S a través de la simulación.

En los últimos años, el ahorro de energía en los sistemas de computación se ha convertido en una necesidad importante. En ordenadores portátiles, la duración de la batería se limita a una cierta cantidad de tiempo, y no desperdiciar energía en ciertas partes haría mas largo el uso del ordenador. Aquí, de nuevo el subsistema de E/S se señala como campo de estudio, ya que las unidades de disco, que son una parte principal del mismo, son uno de los elementos de más consumo de energía debido a su naturaleza mecánica. En los equipos de servidor o de empresa, donde el número de discos aumenta considerablemente, el ahorro de energía puede reducir las necesidades de refrigeración para la disipación de calor y por lo tanto, grandes costes monetarios.

Esta tesis también considera la cuestión del ahorro energético en la unidad de disco, haciendo uso de diversos dispositivos en sistemas de almacenamiento híbridos, que emplean discos de estado sólido (SSD) y unidades de disco. Las SSD y unidades de disco ofrecen diferentes características de potencia, consumiendo las SSDs menos energía que las unidades de disco. En esta tesis se proponen varias técnicas que utilizan los SSD como apoyo a los dispositivos de disco. Se examinan las diversas opciones para la gestión de las SSD y los dispositivos de disco en tales sistemas híbridos, y se muestra que los métodos propuestos ahorran energía y costes monetarios en diversos escenarios. Se ha implementado un simulador compuesto por discos y dispositivos SSD. Esta tesis estudia el diseño y evaluación de los enfoques propuestos con la ayuda de las cargas de trabajo reales.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In many computer systems, the global performance is highly influenced by the I/O subsystem, where the storage devices take an important role. This fact has made that many researchers pay attention to the I/O subsystem as a target of study, and want to improve it. To this end, the usage of real storage devices is demanded, or if it is not possible, storage device models instead.

The energy crisis of the last years and even increasing conscience about the negative effects of energy waste on the climate change, has brought the sustainability both into public attention, industry and scientific scrutiny. Energy demand has been increasing in many computer systems, specially in datacenters and supercomputers, being the I/O subsystem one of the most important parts to take into consideration. This is mainly due to the mechanical nature of disk drives, the most used devices in those centers.

1.1 Motivation

Traditionally, disks have been modeled by means of detailed analytic models based on devices geometry [RW94] or zone splitting [TCG02, dis08], reaching at the emulation level in many cases.

Nevertheless, most of these models are based on out-of-date disks with low storage capacities related to current available disks. If scalable storage systems are being evaluated, needed computing power increases with the number of disks that are included in the simulated system. In that case, using detailed models may impact negatively in the execution time needed to run long simulations. Using such a detailed model is justified only if a simpler model is not able to offer a similar precision.

An alternate approach is to model the behavior of the disk service time, as a stochastic process. The disk, along with its interface and its interconnection mechanisms, may be considered as a phenomenon which has details that are not known. That is a black-box model.

The simplest black-box models are table-based models [JA01], although more elaborate models exist [WAA+04, MWS+07].

Energy saving for computing systems has recently become an important and worrying need. Energy has been increasing in many systems, especially in data centers and supercomputers. As stated by the Green500 [gre11] list, which provides a ranking of the most energy-efficient supercomputers in the world, energy is as significant as performance. Consequently, the performance-per-watt has been established as a new metric to evaluate supercomputers.

In current computer systems, disk drives are one of the most power consuming elements, as they consume about 86% of the total energy consumption in some computing systems [Sym99]. To reduce disk energy consumption, numerous disks provide energy efficient transition modes [DKB95, HLS96]. However, disk idle periods have to be large enough to ensure better power consumption, due to the spinning up/down disk scheme. The desirable goal is to increase disk inter-access times by means of gathering disk accesses with long idle times in the middle.

Solid-state drives (SSDs) provide durable storage through a standard block I/O interface such as SCSI or SATA. SSDs show contrasting features with conventional disk drives. First, they have no mechanical moving parts and hence no positioning delays. Random-access writes in SSD devices can take longer than magnetic disk drives, while reads perform faster [WR10]. Finally, SSDs have lower power consumption ratios. This characteristic makes them optimal to behave as magnetic disk caches thereby reducing disk power consumption.

Applications request previously accessed data when a loop or a function with loops issues I/O requests [BCS+08]. When I/O accessed data are repetitive, write-buffering and prefetching techniques can help to avoid data requests to disk drives [BIC+09].

The work proposed in this thesis starts from the following premises:

- Using detailed storage models may impact negatively in the execution time needed to run long simulations. Black-box approaches need less computational power, and are a good alternative for such cases.
- As soon as a new technological innovation is added, analytical and detailed storage models need to be reworked to include new devices, making difficult to have general models up to date. In black-box approaches, reworking new devices requires the same effort as the required effort by already modeled devices.
- In current computer systems, disk drives are one of the most power consuming elements. This offers a rich set of opportunities for optimization in order to save power in the I/O subsystem.
- SSDs have no mechanical moving parts and lower power consumption ratios. For a power saving goal, they can assist and provide new techniques, helping the disk to stay in lower power states for longer.
- When I/O access patterns are repetitive, write-buffering and prefetching can mask

disk drive accesses, avoiding future accesses to disk drives, and again letting them to stay at lower power states for longer.

1.2 Objectives

The major goal of this thesis is *to propose novel power saving solutions based on SSD devices*. Our solutions include two power saving mechanisms, write-buffering and prefetching. The proposed SSD-based power saving-aware architectures can be used for clusters and supercomputers.

In order to evaluate the proposed power saving solutions, *a new black box model for disk drives is also proposed*. It consists of random variate generators that generate random values, fitting a disk service time distribution.

Additionally, the thesis targets the following objectives:

- **Scalability**. The proposed black-box model targets to use random variate generators that aim to reduce needed computing power by detailed models, when the number of included disks in the simulated system increase.
- **Accuracy**. Our black-box model must offer, if not less, a similar precision as detailed models.
- **Saving power in hybrid storage systems**. The proposed power saving solutions target to use hybrid storage systems. Many existing approaches target only one sort of disk architecture to save energy. This dissertation proposes power saving solutions based on disks and supporting SSD devices.
- **Disk Reliability**. The proposed power saving solutions must be reliable. As to get important energy savings some disks need to be spun down, it is important to take into account the number of times the disks can be switched off and switched on before failures increase.
- **Flash Reliability**. As the proposed power saving architectures use SSD devices, it is important to take into account that their memory internals have a limited number of erase cycles they can cope with, before they become useless.
- **High-Performance**. Although some tradeoffs in performance exist by using power saving techniques, the proposed solutions must offer I/O performances similar to the original (without energy savings) solutions.
- **Economic-Feasibility**. The proposed power saving techniques must reflect, not only reductions in energy consumption, but also in subsequent monetary wastings.

1.3 Structure and Contents

The remainder of this document is structured in the following way.

- Chapter 2 *State of the art* contains the state of the art, and an extensive bibliographic review on the topics related with the thesis.
- Chapter 3 *Black box modeling* outlines a black box model to simulate storage devices service times.
- Chapter 4 *Energy-aware Architectures* presents the design and implementation of a power-saving solution based on SSD devices. Additionally, this chapter is dedicated to present two power saving mechanisms, write-buffering and prefetching.
- Chapter 5 *Evaluation* reports results for both Black Box modeling and Energy-aware approaches.
- Chapter 6 *Final Remarks and Conclusions* contains a summary of this thesis, publications and future plans.

Chapter 2

State of the art

This chapter presents the state of the art related to this dissertation and the background concepts necessary for the understanding of the solution. The material is organized in four subsections:

- Hard disk and Solid state disk basics
- Storage modeling
- Power saving solutions
- Prefetching and caching algorithms

2.1 Hard disk and Solid state disk basics

For years, I/O subsystem has been identified as a bottleneck because its performance improvement has not been occurring at the same speed as in other architecture components. Also, in the last few years, I/O subsystem components have been pointed out as one of the most power consuming parts of the whole system. In both cases, hard disk drives nature, which lays behind the I/O subsystem, is the main thing to blame. Our goal is to provide both a new simulation approach that allows to study performance improvement and also several approaches for power saving in disk drives. For power saving, we use solid sate disks. First of all, it is important to know how both devices work.

2.1.1 Hard Disk basics

Hard disks (HDs) has been for years the most usual devices for secondary storage of data in general computers. Nowadays, they still are due to their high capacities, low prices and improvement in average access times.

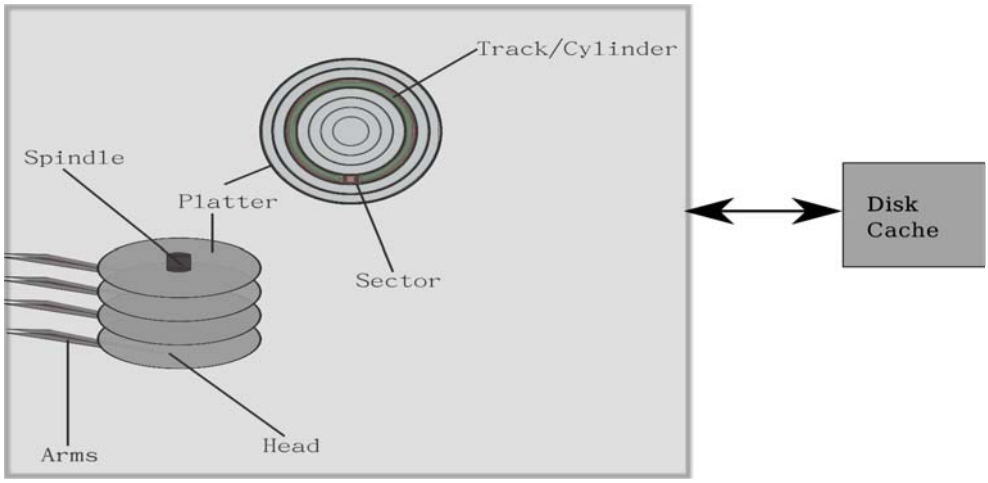


Figure 2.1: Parts of a common Hard Disk

Usual disk drives have a spindle that holds the platters, which are circular disks where data are stored. The read-and-write heads read/write the data from/to the platters. An arm moves the heads letting them to access the entire platter.

Usually, a platter has two surfaces on which data can be stored, and hence, two heads per platter. Each surface is organized in concentric strips (called tracks) that have sectors. A sector is the smallest addressable unit by a hard drive. Its size is commonly 512 bytes.

The set of the parallel tracks on each platter surface is called cylinder. A typical disk will have as many cylinders as tracks per platter surface. Figure 2.1 gives an overview of a disk drive most important parts.

The disk access time is related to its mechanical nature. It can be divided into 3 parts: Seek time, rotational latency and data transfer time. Seek time is how much it takes to the read/write head to reach the track that contains the data to be accessed. Once the desired track is reached, rotational latency is the delay that takes to the head being under the required sector. The data transfer time is how much it takes to the head to read the requested data. The more data to read, the longer it takes.

An access to the platters usually takes several milliseconds. To alliviate this effect, disk drives have internal caches which service requests much faster than servicing them directly from the platters. When a read request comes to the disk drive, the disk cache is asked first to see if it has the requested data. If the requested data are in the disk cache, they are serviced from it, and an access to the platters is avoided. On the contrary, if the requested data are not in the disk cache, the requested data are serviced from the disk platters, and then, saved into the cache. One of the uses of the disk drive read cache is prefetching. When it is activated, the cache loads data that have not been accessed yet, but they are likely to be accessed. When a write request comes to the disk drive, and the caching is enabled, the written data are directly recorded in the disk buffer, thus, improving performance. Those data are usually later written to the platters.

Disk drives can be accessed over several types of interfaces like IDE, SATA, SCSI,

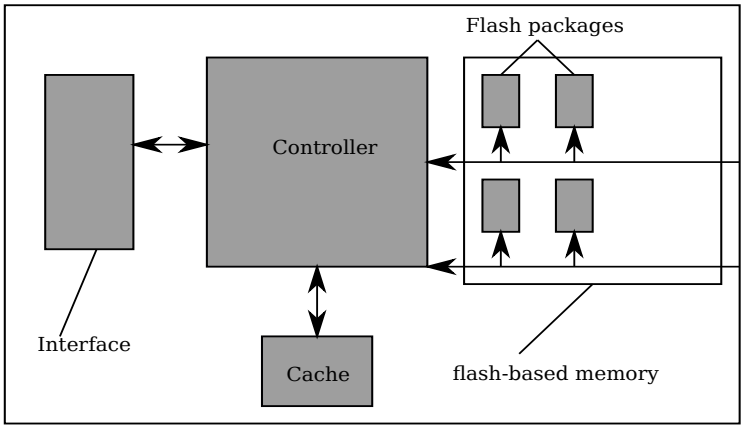


Figure 2.2: Parts of a common Solid State Disk.

SAS, and Fibre Channel. Another interfaces like IEEE 1394 and USB use bridge circuitry to communicate with the disk drive.

With an aim toward saving power, current disks have several power states: *active*, *idle* and *standby*. A disk is in active state when it is reading or writing data. When a disk is not doing anything, but is still spinning, its state is idle. When a disk is not doing anything and its platters do not spin either, its state is standby. The highest power consumption occurs at the active state. Idle state consumes less power than the active state, and standby state consumes much less power than the previous two states. The disk spins down to the standby state when the disk has been in the idle state for a certain period and it is predicted that the state change is worth it. When a I/O request need to be serviced by a disk that is in the standby state, it has to spin up to the active state, and then service the I/O request. Spinning up a disk drive takes several seconds and energy costs.

Every time a disk spins down and spins up, the heads and the spindle motor wear. That is why, manufacturers specify a maximum number of start/stop cycles that a disk drive can stand without producing errors. For desktop drives, start/stop cycles are around 50,000 and for laptop drives, around 300,000.

The cost of a 350 GB SATA disk drive is around (US)$50, which is around (US)$0.14 per GB [SPBW10].

2.1.2 Solid State Disk basics

Solid-state disks (SSDs) are devices that use flash-based memory for secondary storage of data in general computers. They are light, shock-resistant, silent, and consume less power, because they have no mechanical parts.

The main parts of a SSD are the interface, the flash-based memory, the controller, and the cache. Figure 2.2 gives an overview of a solid state disk most important parts.

The interface allows the SSD to connect with a host to behave in a similar way a hard disk drive would do. They can be: SATA, SAS, PCI Express, Fibre Channel, USB, IDE,

and SCSI.

The flash-based memory is usually NAND type because of its low cost. It is composed of several flash packages. Each package has one or more memory chips. Each chip is composed from multiple blocks and each block contains several pages. Usually, a block size is 16−256KB and a page size is 0.5−4KB. Data are read/written from/to in page units. It takes about 20−25μsec to read a page and 200−300μsec to write a page [Sam03, Sam05]. Before a page can be reused it must be erased first. Erasing is at block granularity, that is why it takes more time, around 1.5−2msec.

The controller has the needed circuitry to connect the flash-based memory with the host computer. Among other things, it incorporates a complex module called Flash Translation Layer (FTL) [Int98, M-S98]. One of its important roles is to maximize the parallelism that the multiple flash chips can provide. Multiple chips can be accessed in a interleaved fashion, improving performance of SSDs as RAID would do with disk drives. Other important role of the FTL is to map sectors to the flash-based memory. Several approaches exist, like page mapping and block mapping. In Nand flash, the usual approach is the block mapping. Hybrid approaches also exit [LPC+07, YNS+08].

SSDs have also several chips of cache, that are similar to the cache in disk drives. They help to achieve better performance.

As SSDs do not present mechanical parts, they just have two states of power: *active* and *idle*. A SSD is in active state when it is reading or writing data. When the SSD is not doing anything, its state is idle.

SSDs can be written a limited number of times. As to rewrite a specific block, it must be erased first, what is really limited is the number of times that a block can be erased. When the number of maximum erasures is reached, the SSD's blocks become unerasable but are still readable. There are two kind of NAND flash SSDs: SLC (single level cell) and MLC (multi-level cell). SLC provide only one bit per cell, while MLC provide several bits per cell, increasing the likelihood for errors. This increased likelihood for error also limits the number of times that a specific block can be written, which is around 10,000 times in MLC flash and 100,000 in SLC flash.

The cost of a 128 GB SLC based SSD is around (US)$1,200, which is around (US)$9.2 per GB [SPBW10].

2.2 Storage modeling

A traditional approach for storage devices simulation has been found on analytical models. They use mathematical equations to describe the disk behavior. Many of those ideas were incorporated into simulators, which also simulated in a very detailed way many of the characteristics of the drives, reaching the emulation level in many cases. So, both approaches require a vast knowledge of the device internals to be constructed. An alternate approach is to model the behavior of the disk as it was a black box. The disk, along with its interface and its interconnection mechanisms may be considered as a phenomenon which has details that are not known.

2.2.1 Analytical Models

Analytical models use mathematical equations to describe storage characteristics. Unlike simulators, those descriptions are a sort of summaries of the way some storage parts work. For this reason, they are very fast. However, for some parts of storage devices, it is not easy to reach an equation that describes them.

Disk scheduling algorithms were proposed to reduce disk arms movements, specially during bursty arrival times, treating the disk requests in an order that they have to wait the minimum time as possible. The classic most common scheduling algorithms are: *First Come First Served* (FCFS), Sortest Seek Time First (SSTF), and SCAN. In [JW92, WGP94] new scheduling algorithms, which improved the previous ones, were proposed and studied to be incorporated into other analytical/detailed models or real devices.

The authors of the work [SMW98] described a model composed of several independent models. Each model contains formulas for an specific part of the disk drive, like queueing, caching and disk drive mechanisms. The input to the global model is a workload which is transformed through all the modules until getting a service time. The authors also developed as a side-effect, a read-ahead a data reordering prediction method which takes into account the performance effects of sequential disk acceses.

In [TCG02] an analytical model which analyzes several disk technologies is proposed. Formulas for modeling optical disks with concentric tracks are provided. For magnetic disks, formulas for seek and rotational delays are derived, taking into account both variable and constant rotation speeds. Modeling the new introduced zoned disks was the main contribution of the work.

A description of an analytical throughput model for disk arrays is provided in [UAM01]. As in [SMW98], it is very modular, and can be decomposed to describe the different parts of disk arrays. Among them, modules for disks, buses, and controllers are incorporated. The whole model is compared and validated with measurements taken on real disk arrays. It also incorporates several modern optimizations, which are common in current disk arrays. Its modular design makes the inclusion of new modules easier.

An analytical model for the service time in disk arrays is proposed in [KT91]. When a single request accesses to an array of disks, it can be split into several requests, being the service time, the maximum of the sub-requests' service times. The model provides simple expressions for this.

The work proposed in [YM95] describes a new stripping strategy for disk arrays, and a new disk queuing policy. An analytical model is derived from this, by treating individual disks as independent, and using each one, the M/G/1 queuing model.

Another analytical model for disk arrays is proposed in [VMQ03]. It gives an idea of the performance in disk arrays under synchronous I/O workloads. The model is validated against a real disk array. The model also incorporates many of the typical characteristics included in modern disk arrays, such as, read-ahead and write-back caching, disk scheduling, request coalescing, and redundancy in the array. Also, effects of sequentiality and concurrency are modeled. One drawback of the model is that it has been validated only for workloads containing just reads, or just writes.

As was previously said, memory cells in SSDs can be written a limited number of times. So, an smart management of the internal cells of these devices can extend considerably their life. A mathematical model for flash memory management problems is defined in [BAT06]. Also, several wear-leveling algorithms are analyzed and compared.

An analytical model for the contribution of garbage collection to write amplification in SSDs is proposed in [HEH+09]. Write amplification includes the the more internal writes that have to be done in SSDs, every time they are written. Seeing the results from simulations, it can be concluded that wear leveling is an important key to extend SSDs' life.

The authors of the work [ABJ+09] proposed two analytical models, describing characteristics of SSDs. The first one, is a general flash model, while the other, a unit-flash model. Both models capture different characteristics from flash drives and are useful for designing new models.

The work proposed in [BD11] describes another analytical model to evaluate the performance of two different types of FTLs, in SSDs. Unlike other similar previous proposed models, it uses real workloads. The model was an extension of the model proposed in [HEH+09].

2.2.2 Simulators

Simulators model devices in a very detailed way, in a way that developers of simulators need to understand well the device internals. Simulators can reach in some cases, the emulation level. This characteristic makes them very accurate, but if scalable storage systems need to be evaluated, computation power increases with the number of devices that are included in the simulated system. In that case, using detailed simulators may impact negatively in the execution time when running long simulations.

DiskSim [dis08] is a storage subsystem simulator that includes modules to simulate caches, buses, controllers, device drivers and disk drives. The disk drive, which is very detailed in this simulator, has been validated against a number of disks and even improve other previous simulators [RW94]. The rest of the components (buses, adapters, controllers, and drivers) have not been validated [VMQ03]. DiskSim uses many parameters which are extracted from disks using semi-automated algorithms [WGPW95], or by means of the DIXtrac disk characterization tool [SG00, SGLG02, BS02]. To give an idea of the complexity of the simulator, it is enough to note that DIXtract extracts over 100 performance critical parameters. However, DIXtract can only extract parameters from SCSI disks.

Pantheon [Wil95] is another I/O subsystem simulator that initially was used for simulating parallel disk arrays. It simulates components like disks, tapes and array controllers. Here, again, only disk modules have been properly validated [VMQ03]. It was extended to simulate both uniprocesors and parallel systems.

RAIDFrame [IGHZ96] was also constructed to evaluate arrays of disks architectures. It is a software RAID controller which can also be used as a stand-alone discrete-event simulator for disk arrays [VMQ03].

FlashSim [KTGU09] aims to evaluate storage systems that employ SSDs. It can be used with different FTL schemes and it can be integrated easily with DiskSim. Another

simulator from Microsoft [APW+08] also simulates SSDs and it is very integrated with Disksim. It implements interleaving methods available in common SSD devices but it only lets using one FTL scheme. In [LBP+09] another SSD simulator was developed but it does not allow to test different FTL schemes and cannot be integrated with Disksim.

2.2.3 Black-box models

Black box models assume that almost no knowledge of the storage device is included. Devices and all their characteristics are considered as black-boxes where the internal details are not known. Among other advantages, we can say that they are usually fast, because they are not very detailed. They are easy to construct and to update, as it is not needed to know the internals of the devices and new technologies; and they can be accurate.

The simplest black-box models are table-based models [JA01]. They just relate, in memory tables, information of input data from workloads to output data from real service times. Input data from workloads can be request sizes or request types. Output data are the resulting service times from replaying traces on the real devices. The bigger the tables, the more accurate the models. Missing data in the tables are just interpolated from the most similar characteristics on the tables. Accuracy can be improved by adding new information to the tables. However, this solution is not scalable, because, the more information is added to the tables, the slower the searching in tables, and the slower the prediction of service times.

In [WAA+04] an enhancement of the work in [JA01] is proposed. The authors do not use tables, but applicate CART (Classification And Regression Tree) [BFOS84] modeling to storage devices. In CART models goal/output data are organized in trees whose leaves are accessed by using information from input, like request sizes or request types, or other characteristics from workloads. They propose two approaches: One that predicts per-request response times, and another that predicts aggregated response times. The per-request approach predicts response times just taking into account the characteristics of that request. The aggregated approach predicts by taking into account characteristics from the whole workload. The per-request predictor is also more computationally demanding than the aggregated predictor. On the other hand, the per-request predictor can achieve better accuracy. Evaluations of the model do not use real disks. Instead, instances of the validated DiskSim simulator are used. Unlike this, the solution proposed in this thesis use real disks for validation, and also provides comparisons with instances of DiskSim, in terms of accuracy and performance.

The authors of the work [YUK06] evaluate the accuracy of a black box model, for hard disk drives, based on regression trees. They use an open source algorithm called GUIDE [Loh02]. This algorithm employs Chi-Square analysis of residuals, and eliminates the bias in variable selection. Evaluations include two kind of environments: single workload environment, and multiple workloads environment. The single workload environment simulates a single workload stream, while the multiple workloads environment simulates an scenario with multiple streams. Authors conclude that the multiple workloads scenario is more difficult to predict.

In [MWS+07] the performance of a storage device is modeled by tunning the perfor-

mance, resource utilization and workload characteristics of other device, in which the first device is based. Similar or not similar devices may be related in some ways: They may have similar hardware characteristics or not, they behave similar or not under the same workloads, etc. The main goal of the proposed method is to know the relationship between the performance's devices, in order to construct a derived model from an already modeled device. The method begins with an absolute black box model that is constructed by applying CART [BFOS84] modeling. One drawback from this model is the fact that, if the absolute model is not accurate, errors can be transmitted to the derived model, making it less accurate.

The authors of the work [LH10, HLST11] use regression trees, as in [YUK06], for black-box modeling of SSD devices. They proposed two approaches: A basic model and a extended model. Both models take as input workload characteristics and obtain predictions of performance. In the basic model, the workload is characterized by the percentage of writes/reads, requests sizes, number of pending previous requests, and the percentage of random requests. In the extended model, workload characteristics are the same but are split by taking into account if they are read or writes, due to the asymmetric performance of SSDs. Also, two other different patterns from random are studied, like sequential and strided. However, evaluations do not use real traces. They just use synthetic traces with the previously mentioned, predefined patterns.

In [LZY+11] a black box energy consumption model for disk arrays is proposed. Here, a disk array, altogether with a controller and several disks, is considered as a black box, whose energy consumption can be measured.

The solution proposed in Chapter 3 is a new method to construct black box models for disk drives. As in DiskSim [dis08], it provides a tool to extract disk characteristics. Unlike DIXtrac [SG00, SGLG02, BS02], which only characterizes SCSI disk drives, the proposed tool can be applied to any disk, with any kind of interface. The black box model proposed on this thesis is based on probabilistic distributions. This fact makes it very fast at predicting stage. In order to demonstrate this, it is compared with DiskSim in terms of performance. It is also evaluated using real traces, and synthetic traces from a widely known benchmark, SPC-1[Cou06]. The proposed model is validated against a real disk. It is also compared with DiskSim in terms of accuracy. Results from evaluations show that the proposed method can be as accurate as DiskSim.

2.3 Power saving solutions

Power saving solutions described in this section aim to save power in computer systems. Hard disk drives are identified as one of the most power consuming elements of the computer system, and especially, of the I/O subsystem context. Traditional power saving studies have proposed techniques for saving power in laptop/desktop disk drives. Recently, power saving in enterprise storage systems has also generated a lot of research.

2.3.1 Laptop/Desktop oriented power management

Portable computers can only work for several hours, in an autonomous way, before finishing off a single battery charge. Several components are identified as participants of this energy wasting. Screen has been identified as the most power consuming element, followed by the hard disk drive [LKHA94]. Most techniques take advantage of some inactivity periods to stop the components and save energy, when they are not used. This technique is the most applied in the hard disk drive context. So the goal here is mainly to save battery energy by stopping the hard disk drive, and thus, enlarging the amount of time the computer can operate.

2.3.1.1 Spin Down Policies

As previously said, disk drives can save power by being stopped. Stopping a disk means putting it in a lower power-mode. Disk drives can be in three power modes: *active* (the platters are spinning and the heads are servicing requests), *idle* (the platters are spinning but the heads not servicing requests) and *standby* (the platters are not spinning and the heads are parked, saving energy). When a disk has been in the idle mode for a specific period of time, it spins down to the standby mode. Once a request comes to the disk, it spins up to the active mode to service the request. As spinning down and spinning up processes consume time and energy, the disk should stay in the standby state for at least a minimum period of time, for that changes to be worth it and for the disk to save energy. Spin down policies determine when to spin down a disk to save energy.

The authors of the work [LKHA94] provide an analysis of the tradeoffs and benefits of spinning down a disk drive as a power reduction technique. First of all, the most suitable spin down delay (the length of time the disk waits for further activity before going to the standby state) is found out. It is demonstrated that a spindown delay of 2 seconds results in the most energy savings. Second, the tradeoff between energy consumption and user delay is analyzed for the previous 2 second spin down delay. The results show that the user will have to wait for 15-30 seconds per hour, which are 2 second disk spinups in around 8-15 times per hour. Next, the use of a disk cache is analyzed for energy savings. It is demonstrated that using a one megabyte cache is sufficient to achieve most of the energy benefits of disk caching, and a bigger size cache will not yield any additional savings. Moreover, delaying 30 seconds in writing dirty blocks from cache to disk will provide additional energy savings, and more than that value will not provide more savings. Another analyzed question is the fact that this technique provides increased disk spinups and makes bigger the wear on the disk-head interface. This problem is solved with the next generation of disk drives which lets a longer continuous use. Finally, as names and attributes in the file system are frequently accessed, also the use of a name and attribute cache is proposed. More energy savings are obtained with the use of it.

In [DKB95] some adaptive spin-down policies are proposed. The user has to define a level of acceptability which is a metric that shows the number of undesirable spin-ups are accepted by him/her. If the user defines a high level of acceptability, a lot of spin-ups will take place by trading-off performance and decreasing energy. On the contrary, if the user defines a low level of acceptability, less undesirable spin-ups will take place, energy

savings will be decreased and the performance will be better. Threshold adjustment takes place during an adaptive policy, either when an unacceptable spin-up occurs or when other information suggests the need to change the threshold. Adaptive spin-down is compared with fixed threshold policies. The results show that:

- sometimes adaptive spin-down policies eliminate a big part of all undesirable spin-ups with small increases in energy consumption,
- other times the undesirable spin-ups are the same obtained by varying fixed threshold policies, and
- sometimes the results are worse than using a fixed threshold policy.

It is important to note that in some cases the effectiveness of the adaptive spin-down policies depends on the trace and disk used.

In [HLS96] the problem of when to spin down the disk in mobile computers is treated by using a machine learning algorithm, *The share algorithm*. It receives as input a set of experts which make predictions. The algorithm combines the predictions of the experts and assigns one weight per expert. That weight represents the quality of each expert predictions. The weights of the experts are continuously being updated. Results show that *The share algorithm* is better than other practical algorithms, usually implemented, such as:

- the fixed spin-down delay algorithm, which picks one fixed spindown delay as value and spins down after the disk has remained idle for that period,
- the fixed spin-down delay equal to the spin down cost of the disk algorithm, or
- randomized algorithms, which select spin-downs delay values from some distributions.

The implementation of the algorithm is quite efficient, and could be implemented on a disk controller.

2.3.1.2 Energy-aware Prefeching and caching techniques

Choosing well when to spin down a disk helps to save energy. When a disk has been in the idle state for a certain period of time (set by the spin-down policy), it goes to the standby state. Once a request comes to the disk, it has to spin up to service it, regardless of the spindown policy. Prefetching and caching techniques, which reduce the number of disk accesses, become a very useful support for the spin-down policies to save even more energy. When a request comes to the disk, and it is in the standby state, if the request is in the cache, an spin-up can be avoided, and disk idle times can be longer, thus saving more energy. The main goal of these approaches is to maximize the disk idle times by using caching techniques so as to keep the disk in the standby power mode as long as possible.

Prefetching caches are usually used to hide disk latencies, but it does not usually reduce the power consumption of the disks. However, the authors of the work [PS04] proposed an epoch-based prefetching algorithm into the memory management mechanisms

of the operating system, for power saving goals. Each epoch consists on two phases: idle phase and active phase. During the idle phase the disk is in a lower-power mode until a new prefetching cycle has to be initiated, a miss takes place or the system is low on memory resources. In that idle phase acceses to each active file are monitorized by the operating system by using daemons that try to predict the next miss rate for each file. During the active phase the prefetching is carried out using the information monitorized by the daemons and the length of the upcoming idle phase is predicted. Moreover, accesses of concurrently running applications are coordinated for them to arrive the disk roughly at the same time. The main goal of using this prefecthing algorithm is to maximize the disk idle time by increasing the burstiness of I/O patterns. Experimental results show that the bigger is the size of the memory used by the system, the better are the energy savings. The energy savings are up to 60-80%.

In [CZ08] two caching policies are described in order to save energy. Both policies try to maximize the disk idle time by reshaping the disk access pattern to a clustered one. The first one, HC-Burst, is a replacement policy that chooses more clustered accessed blocks and more unlikely to be accessed blocks for replacement by identifying burstiness in previous periods of time. Clustered accessed blocks do not break idle intervals, because all the blocks are accessed in short intervals of time. The second one, PC-Burst, is a replacement policy that tries to predict disk accesses in the future. By using this information some disk accesses can be evicted in order not to break long idle times. The energy savings are up to 35% using a real implementation and the performance loss is minimal.

Laptop Mode[San04] is a setting that comes with the Linux Kernel and lets changing the way one wants to distribute disk I/O requests over time. Usually, Linux distribute disk I/O in a smoothly manner, not letting the disk to spin down for long periods of time. In a laptop, in order to extend its battery life, it would be beneficial to concentrate requests into shorts periods of time, with long intervals of inactivity between them. So, whenever Laptop mode is activated it is possible to extend periods of inactivity up to ten minutes. During the activity periods read-ahead can be performed up to 4MB, and this evicts some spinups. Moreover, writes can remain in memory until the inactivity periods are over or the memory is out.

In [TLY06] an energy efficient virtual memory system with flash memory as the secondary storage is used. Two different techniques are proposed: subpaging and storage cache management. The subpaging technique consists on partitioning a virtual memory page in several flash pages, and only dirty subpages are written to flash memory whenever a page faults. In the storage cache management, only writes are cached in SRAM and two approaches are showed: the first one keeps in cache the most frequently and the most recently used pages (TF policy). The second one takes into account the garbage collection overhead problem, trying to allocate data accessed close in time to the same flash block so a victim block with a small amount of live pages could be found whenever garbage collection is triggered (TF-Locality). The subpaging technique energy savings for a flash memory are up to 20% on the average, and up to 24% on multiprogramming workloads. The TF policy energy savings are up to 19.5%. The two techniques together get energy savings of up to 35.6% on average for a flash memory.

2.3.1.3 External Caching

External caching differentiates from in-memory caching in the fact that in external caching a non-volatile storage device is used [UGB+08]. Therefore, caching or prefeching policies are applied to it to save energy in the disk drive. Using non-volatile storage devices like NAND flash devices as caches for disk drives, has several advantages over buffer memories. First, they have high densities and thus can store lots of data. Second, they are non-volatile, which means that when a power cut occurs, data do not have to be lost. Third, they have low power characteristics. However, NAND flash devices are not still as fast as in-memory caches.

EXCES [UGB+08] is a dynamically loadable Linux Kernel module that operates between the filesystem and I/O scheduler layers in the storage stack. It is composed of five components: page access tracker, indirector, reconfiguration trigger, reconfiguration planner and reconfigurator. The module tries to redirect as many requests as possible to a flash-based storage device and to keep the disk sleeping as long as possible. So the flash based storage device acts like a sort for cache for the disk. The page access tracker catches every request and maintains information about the popularity of the requested pages. The most popular pages are the most recently and the most frequently accessed. The indirector component redirects the request either to the flash-based storage device or to the disk. The request is redirected to the disk whenever there is no space in the flash-based storage device and the requested page is not in the flash-based storage device, if it is a write. If it is a read and the requested page is not in the flash-based storage device it is directly redirected to the disk. Then, the reconfiguration trigger is invoked and it decides if a reconfiguration operation is needed. If so, the reconfiguration planner uses the popularity pages information to create a reconfiguration plan to move some pages from the flash-based storage device and vice versa. Then, the reconfiguration is invoked to carry out the plan created by the reconfiguration planner. Energy savings with this approach are found in the range 2-14% and important reductions in performance are noted more in write-intensive workloads due to random writes to the flash-based storage device that are the least efficient, both in performance an power consumption [BITW07].

In [KRM08] a Flash based disk cache is proposed. With this approach, Flash is used as a way to address the latency gap between the hard disk drive and DRAM while saving power in the server domain environment. The Flash based disk cache is split into two regions, one for reading and the other for writing. Moreover, a programmable Flash memory controller is proposed in order to improve Flash cell reliability and extend memory lifetime. Four tables are hosted in the primary disk cache (DRAM) to carry out all the block and page management (*Flash Cache hash table*, *Flash page status table*, *Flash block status table*, *Flash global status table*). They split the Flash based disk cache into a read and write cache in order that less blocks are candidates for garbage collection. This reduces Flash reads, writes and erases compared to a unified Flash based disk cache. When a read is performed, the DRAM is searched first. If there is a hit, the content is accessed. On the other hand, if a miss occurs, the Flash based disk cache is searched. If the requested content is found, it is accessed, if not, the disk is accessed and the content is copied to the DRAM and the read cache in Flash. When a write is performed, the DRAM is accessed and then the page accessed in DRAM is written to Flash. If the page is in the write cache, it is updated, and

if it is in read cache, it is invalidated and copied to the write cache. In order to improve Flash cell reliability and extend memory lifetime, either ECC strength can be increased for a block or that block can be switched from MLC to SCL mode.

Both in [BBL06], and in [CCJZ06], a unique flash drive in considered as a cache for a single disk drive, in the laptop/desktop environment. In [BBL06], a very simple sequential prefetching algorithm is proposed. In [CCJZ06], a more complex prefetching algorithm is proposed, which uses information from the kernel. Here, the kernel provides hints of file access patterns. Also, in both cases, caching techniques are applied.

2.3.1.4 Hybrid disks

Hybrid disks locate some flash memory next to conventional disks. Usually, the disk part stays at the standby mode. Writes are written to the flash part. Reads are read from the flash part when they can be served by it. When not, more data apart from the needed are read from the hard disk to the flash part, aiming to serve future requests. When the flash part is almost full, hard disk part spins up to move the written data to it from the flash part. Also, when read data cannot be served by the flash part, hard disk part spins up to serve them.

An operating system can take advantage of the flash part to achieve random acceses at more speed and also boot times, because flash memory presents constant access times for reads.

Hybrid disks are more expensive than the conventional hard disk drives. Also, they provide less performance when data has to be read from the hard disk part and it is in the standby mode. This is because spinning up the hard disk part takes time, and in some cases, several seconds. Hybrid disks presents also lower performance for write requests, as a consequence of the flash memory nature.

Life time of hybrid disks is more limited than in the conventional hard disk drives. That is because both the hard disk drive part and the flash part have reduced life times. Hard disk drive parts have a limited number of spin down/up cicles. Flash parts have also a limited number of erasure cycles per block.

Hybrid disk are noisier than conventional disk drives. This is because spinning up/down processes are noisy.

In [BBL07] a hybrid disk is used to save power in desktop/laptop computers by using several techniques:

- Artificial Idle Periods: It is an improvement in a spin-down algorithm which considers I/O type when the disk idle time is computed. Idle times are recorded by taking into account the time since the last read request, and not only since the last request. Most of write request can be served by the flash part, avoiding spinning up the hard disk part. By doing this, idle periods are increased, and also, energy savings.

- Read-Miss Cache: It is used to host potential read-misses altogether with the subsequent sequential reads, in the flash part. Its goal is to reduce spin-ups caused by future read operations.

- Anticipatory Spin-up: It aims to reduce the time it takes for the disk part to spin-up. It wakes the disk up in advance of the fact that a request that cannot be serviced by the flash part, arrives.

- Write Throttling: Controls the amount of data written to the flash part while the disk is not being used, so that it can last for certain time.

2.3.2 Enterprise Storage Systems oriented power management

Saving energy in enterprise storage systems has become an urgent need, because it is usual to have a great number of disk drives, working at the same time. This fact increases power use, and also greater cooling requirements for heat dissipation, reduced computational density, and higher monetary and operating costs. For a usual data center, storage accounts for 27% of energy consumption [WOQ+07].

2.3.2.1 Multi-speed drives

Some approaches use multi-speed disks, which are be able to change their speed while spinning. Slower rotation speeds waste less energy than higher rotation speeds. According to the incoming workload, multi-speed disks change their rotation speed, to save energy. The lighter the load, the slower the rotation speed, and vice-versa. Slow rotation speeds can still service requests by providing lower energy consumptions. However, such disks are still inaccessible in the market.

In [CPB03] four approaches are proposed to save energy in the network servers context. The first one consists in powering disks down during periods of idleness. This technique is not very appropiate for network servers due to the lack of idleness in the disk subsystem. The second one consists in replacing a high performance disk with several lower power disks. It is demonstrated that the replacement works well for storage, performance and reliability requirements, but not for energy. The third approach consists in using one high-performance disk and a lower-power disk. Both disks should have the same data. During times of high load the high performance disk is used, and during periods of lower load the lower-power disk is used. Sometimes it is necessary to update blocks of data from one disk to the other one. This technique is only appropiate for not realistic disk demands that the lower-power disk is able to cope with. The last approach, multi-speed disks, uses several-speed disks. The higher the load, the higher the disk speed. When the disk load becomes lighter than 80% of the disk throughput of a low speed, the disk spins down to the low speed mode; if the load is heavier than the same threshold, the disk spins up to the high speed. This technique works well in performance and obtains energy savings between 22% and 15%.

DRPM [GSKF03] is a mechanism to dynamically modulate the disk RPM speed. To see the potential of the strategy either the RPM transition costs and power characteristics are modeled. A detailed study with different workloads and each one with different characteristics and level of burstiness demonstrates that DRPM outperforms traditional power management techniques [LKHA94, DKB95] in intermediate cases when the inter-

arrival times are not either too small or too large. The reason is the fact that DRPM finds more opportunities to transition to lower power modes, which may not be long enough for traditional power management schemes. Moreover, DRPM provides an heuristic for a disk controller array to decide at the end of each n-requests window whether to command the disks to go to full speed when the average response time is larger than an upper tolerance level, to go to a lower speed when the average response time is lower than a lower tolerance level, and to keep their speeds when the response time is within both tolerance levels. It is important to indicate the size of the window (in number or requests) and the both tolerance levels, always taking into account the power-performance tradeoffs.

Hibernator [ZCT+05] is an approach to save energy, getting at the same time good performance goals. A disk array is used having each disk several speeds. A coarse grained algorithm to decide at which speed will work each disk, in each epoch is proposed, always taking into account either the average response time, and the energy consumption for each disk in the previous epoch. Moreover, some layouts and ways to migrate data to the appropiate disks are proposed. Among them, the most appropiate is *randomized shuffling* which lets add disks to tiers (groups of disks which spin at the same speed) and remove disks from tiers as necessary. The energy savings are up to 65%.

SBPM [SGS08] is a mechanism that dynamically modulates several knobs in the storage system, such us the disk RPM and the VCM speed to adapt the storage system to workload conditions and save energy. It is demonstrated that this technique outperforms *DRPM* in all kind of characteristics. For example, *DRPM* consumes more energy and its performance is worse than *SBPM*, since the assumptions about the time taken to transition between RPMs in *DRPM* is in the millisecond range while in *SBPM* is in the second range as happens in real multi-RPM drives. Moreover, the *DRPM* policy introduces more RPM oscillations than *SBPM*, which is able to better balance the system by using both the SPM and VCM knobs.

2.3.2.2 Energy-efficient RAIDs

Redundant Arrays of Independent Disks (RAID) provide reliability and high performance by putting several disks altogether and treating them like an unique device. Saving energy is exploited at RAID level as well.

PARAID [WOQ+07] is a new layer that can be integrated in a RAID 5 configuration. It can vary the number of power-on disks depending on the incoming load. Data are distributed among more disks for high workloads. When workloads are lower, data are concentrated in few active disks. It can reduce power consumption by up to 34% and can keep a reasonable level of performance and reliability.

MAID [CG02] moves infrequently used data to unused disks and take a role similar to tapes because in this environment, a great deal of data are written and never accessed again. This configuration achieves energy savings of 85%.

RIMAC [WYZ08] is a redundancy based I/O cache architecture to reduce energy consumption and improve performance. The main contribution of RIMAC is that it solves the *passive spin-up problem*. In this problem, if a disk has been spun down for a not-big-enough period, the wasting in performance and power can be more than the one from

keeping the disk in the active state. In RIMAC a large storage cache to store data stripe units is used, while a smaller RAID controller cache is used to save parity units. In this approach, only one disk can be spun down. The authors of RIMAC develop two read request transformation schemes (TRC and TRD) and a power-aware write request transformation policy for parity updates. Both policies let reads to the non-active disk, to be serviced by the storage cache, or by calculating an on-the-fly XOR block with data and parity from the active disks, if those data are not hits on the storage and controllers caches. The authors also developed a second-chance parity cache replacement algorithm to improve the success rate of power-aware request transformation policies such as TRC and TRD. Results show that RIMAC saves energy up to 18% and reduces response times up to 34%.

EERAID [LW04] use non-volatile caches to host parity or data blocks. When a request to a block needs to be service and that block can be calculated both from the blocks in the active disks and from the blocks in the cache, there is no need to wake up a power off disk.

2.3.2.3 Data migration across drives

Some techniques create enough inactivity in a number of disks by migrating their data to another disks.

Write-off Loading [NDR08] is a new technique that allows write requests on spun-down disks to be temporarily redirected to persistent storage elsewhere in the data center. This increases energy savings to 45%-60%. The used hardware to host off-loaded blocks is the end of each existing volume. Each volume has its own manager who decides when to spin disks up or down and where to off-load writes. The places where the blocks are off-loaded in other machines are called loggers. The block-level traces from an enterprise data center were analyzed and an important conclusion of that was the fact that there is enough idle time for some volumes to spin down without removing the writes from the traces. So the both cases show there is idle time for the volumes to spin down and save energy.

In [NTD+09] an analysis of tradeoffs and benefits to know if replacing disks by SSDs is worth it, in the enterprise environment, is presented. Several possibilities are considered: keeping the current configuration based only in disk drives, combining SSDs with disk drives, and replacing totally disk drives by SSDs. Analyzed figures, in terms of prices and performance, show that the high prices of SSDs do not make them adequate to replace hard disks yet. As a conclusion, at SSDs current prices, combining SSDs with hard disks could be seen as a adequate transition solution.

MAID (*Massive arrays of idle disks*) [CG02] are large storage arrays designed to reduce the energy costs while maintaining performance in supercomputer archival storage environments. In this kind of environments a great deal of data are written and never accessed again, so they have little need of the high performance or increased reliability of conventional RAID systems. The authors provide accurate models for both performance and power. The system is composed of several *cache drives* that remain spinning and several *data drives* which can spin-down following some periods of inactivity. Several configurations for the previous system are analyzed and the results show that the same performance is maintained while achieving energy savings of 85%.

The proposed approach in Chapter 4 redirects data from disks to SSDs, in the scope of large parallel computing environments such as clusters or supercomputers. When doing that redirection, life span of SSDs, and disks are also taken into account. An economic based evaluation of such redirection is described in Chapter 5.

2.3.2.4 Power-aware cache management

PB-LRU [ZSZ04] is a power-aware algorithm which divides the storage cache into separate regions, one for each disk. There is correlation between a disk's energy consumption and its corresponding storage cache partition size. That means that the size assigned to inactive disks will be bigger than the size assigned to active disks, and thus the energy consumed by inactive disks will be less than by active disks.

As this solution uses volatile memory as cache, reliability problems can appear, bringing on data losses.

2.3.2.5 Power-aware File Systems

Saving power is exploited at file system level.

BlueFS [NF04] is a file system that checks the energy features of each device to determine when to access data. This configuration achieves energy savings of 55%.

Conquest-2 file system [RAG+03] stores small files from disks in non-volatile RAM to save energy.

GreenFS [JS08] is a FileSystem that lets many of the disk drives from an enterprise to be kept in the standby state for enough time to save energy. In GreenFS the local hard disks are just backups of the data stored remotely in a GreenFS server. Local hard disks are used when the local network does not work well. In GreenFS a client has also a local flash memory which serves as a buffer that hosts data updates when the remote server is not available. Thus, flash memory can make bigger the local memory cache size and improve performance on the client. GreenFS increases enterprise data reliability, minimizes disk power consumption, noise and its design requires relatively minimal hardware and software modifications in existing infrastructures.

FS2 [HHS05] is a file system based on the Ext2 file system. It contains a runtime component responsible for dynamically reorganizing disk layout. FS2 replicates disk blocks to improve disk I/O performance and energy savings. By preserving replicas adyacency on disk according to the other in which their originals were accessed at runtime, future accesses to blocks will be made significantly faster.

In this thesis, a generic block level solution is proposed, which is transparent to the file system, and other higher levels in the operating system.

2.3.2.6 Power-aware server clusters

In [PBCH01] energy conservation for clusters of PCs is treated. The technique switch cluster nodes on in order to manage high loads efficiently, and off, to conserve energy under more

moderate loads. The main part of this system is an algorithm that carries out decisions by considering both the load on the cluster and the energy and performance involvements of turning nodes off. The algorithm is tested in two different ways: at an application degree, and at an operating system degree. The results are encouraging, demonstrating that the method saves energy in relation to other conventional systems.

The solution proposed on this thesis is just focused on the I/O subsystem, where switching on/off times are less than switching whole I/O nodes.

2.4 Prefetching and Caching algorithms

Prefetching and Caching algorithms are a common practice in storage systems. They mainly have been used to improve performance by avoiding as many I/O requests as possible, to increase throughput and decrease latency. Moreover, in one way or other, they could be used to create bursty access patterns for disk drives, increasing the average length of idle intervals and maximizing utilization when the disk drive is active.

2.4.1 Prefetching algorithms

Prefetching algorithms move data, ahead of time, from the I/O subsystem to the cache. Those moved data are supposedly going to be accessed soon. If some of the moved data happen to be in the cache when accessed, some high service demands that come from extreme head displacements at disk drives, can be avoided. Moreover, when the goal is to save energy, having future accessed data in the cache may prolong idle times sizes, letting disk drives save energy. As was previously said, traditional prefetching algorithms have as an aim to increase throughput and decrease latency. The next paragraphs describe some of them.

The *C-Miner* algorithm [LCZ05] tries to discover block correlations in storage systems by employing a data mining technique called *frequent sequence mining*. In this technique, frequent sequences of accessed blocks are evidences of blocks correlations in storage systems. Knowing which blocks are going to be accessed after accessing a group of previous blocks, the future accessed blocks can be prefetched in the cache. The algorithm has been frequently used for reducing the average I/O response time in storage systems, but no for power saving goals. The algorithm does not really work with sequential workloads because they do not usually happen often enough in the access flow to be caught by C-Miner, and it needs some level of repetition for the block accesses. C-Miner algorithm works assuming that block correlations are stable, and running it only once, the correlations will remain stable for a long time. So, although the algorithm takes about an hour to run for requests from the previous days, it is not necessary to run it constantly in the background, just only once to update block correlations every week.

The *Adaptive Prefetching algorithm in disk controllers* [ZGQ08] uses an adaptive prefetching scheme to optimize the system performance in disk controllers for traces with different data localities. The algorithm has not been used for power saving goals. The algorithm uses on-line measurements of disk transfer times and of inter-page fault rates to

adjust the level of prefetching dynamically, and its performance is evaluated through trace driven simulations using real workloads. The results confirm the effectiveness and efficiency of the adaptive prefetching algorithm.

The algorithm presented in [GAN93] uses past disk accesses to predict future accesses. It presents an adaptive method. The adaptive method consists of an adaptive table which hosts information about past disk accesses. That information is used to predict next probable disk accesses. The information is also continually updated to the next probable disk accesses. The algorithm also adapts and works in multitasking environments, where multiple streams of disk requests are mixed. Several variations of the same algorithm are tested.

In [GM05] sequential prefetching is considered very integrated with cache replacement policies. Also, the caching space is divided to host sequential and random streams and avoid read disk accesses. The authors design a sequential prefetching algorithm, called SARC. It is capable to combine synchronous and asynchronous prefetching. In synchronous prefetching a sequential stream of blocks is brought to the cache when a miss occurs. In asynchronous prefetching, a miss does not have to happen to bring subsequent blocks in advance. It also uses in an adaptive way that separation of sequential and random streams to minimizing the hit ratio and maximizing the average response time. The algorithm is tested in a Shark storage controller.

In [GB07] a distinction of four sequential prefetching algorithms is defined taking into account if they are synchronous or asynchronous. The authors designed a novel algorithm, AMP which applies the theoretical analysis previously done in for the sequential prefetching algorithms.

With an aim to save power, several prefetching algorithms have been previously mentioned, and described in [PS04], [CZ08], [San04], [CCJZ06], and [BBL06].

In this thesis, four prefetching algorithms, with an aim to save power, are proposed. The first three are dynamic, and suitable for sequential-oriented workloads. The other, as *C-Miner* is offline, and tries to extend disk idle times, mainly by prefetching data determined by previous experiments.

2.4.2 Caching algorithms

Caching algorithms decide which data will be replaced whenever a cache is full. As was previously said, traditional caching algorithms have as an aim to increase throughput and decrease latency. The next paragraphs describe some of them.

Belady's algorithm [Bel66] replaces, specifically, data that are not going to be accessed in the longest time. This is an ideal algorithm, and there are no implementations of it. That is because, it is not always possible to know in advance, which data are going to be accessed in the future. It is specially useful to see the effectiveness of other caching algorithms.

The *Least Recently Used (LRU)* algorithm [CD73, Den68] replaces the data in the cache that have not been manipulated for the longest period of time. It is based on the experience that data which have been used in the recent past will probably be used again in the proximate future.

The *Least Frequently Used (LFU)* algorithm replaces the data that are least frequently manipulated. The motivation for this algorithm is that some data are accessed more frequently than others, so reference counts can be used as an estimate of the probability of data being referenced.

The *Random Replacement (RR)* algorithm randomly selects data to replace. It is very fast, and no information about the cache needs to be kept.

The *Multi-Queue (MQ)* [ZPL01] algorithm uses multiple LRU queues. This algorithm demotes data from higher to lower level queues in order to eventually evict data that have been accessed frequently in the past, but have not been accessed for a long time. This algorithm evicts double caching effects that come with using other caching algorithm in the operating system buffer caches.

In [KLW94] the *Segmented LRU (SLRU)* is described. It divides the cache into two segments, making a combination between LRU and MRU. The *Adaptive Replacement Cache (ARC)* [MM03] improves the SLRU algorithm by combining LRU and LFU, and dynamically adjusting the size of the two segments.

With an aim to save power, several caching algorithms have been previously mentioned, and described in [PS04], [CZ08], [CCJZ06], [BBL06], and [UGB+08].

In this thesis, as will be described in Chapter 4, the *LRU* algorithm was used.

2.5 Summary

This chapter have presented a few preliminaries to understand the two problems presented on this thesis, and a complete compendium of solutions for the two analyzed problems.

A preliminary description has detailed the main characteristics of hard disks and Solid State Disks.

For the disk modeling problem, solutions for analytical, detailed, and black box models are presented. Also, the solution proposed on this thesis has been placed among the black box model solutions.

Regarding the power saving problem, solutions both for laptop/desktop and enterprise environments are described. Moreover, the solution proposed on this thesis has been placed among the enterprise environments solutions, in the data migration branch. Also, as it uses prefetching and caching policies, a collection of them is detailed.

Chapter 3

Black box modeling

3.1 Introduction

Simulation is a common technique in performance evaluation of many applications. To obtain realistic results, data access to either a file system or database management system cannot be ignored. However, in any case a key element in the data access is the storage device simulation model.

In general, a storage device model accepts as input the parameters of an application workload (as a flow of device requests) and outputs a prediction of a performance metric. The output performance measurement may be a general performance metric as the average bandwidth, throughput or latency. Such metrics give an idea of the global performance of the device. However, if the device simulation model is to be integrated into a larger model which includes a file system or a database manager a detailed metric as the response time for each device request is needed.

Traditionally, storage devices have been modeled by means of detailed analytic models based on devices geometry [RW94], zone splitting [TCG02, dis08] or the use of of read-ahead caches and request reordering [SMW98], reaching the emulation level in many cases. One drawback of analytic models is that, as soon as new technological innovation is added, the model needs to be reworked to include new devices making difficult to have a general model up to date.

Many of theses characteristics have been incorporated in simulation tools. One widely simulation tool, where many of these ideas have been integrated is DiskSim [dis08]. The simulator uses many parameters which are extracted from disks using semi-automated algorithms [WGPW95], or by means of the DIXtrac disk characterization tool [SG00, SGLG02, BS02]. To give an idea of the complexity of the model, it is enough to note that DIXtract extracts over 100 performance critical parameters.

There are system wide simulations where scalability is an important issue. This is the

case in large clusters design [NFG+09], evaluation of peer to peer or volunteer computing models, Grid storage infrastructure or content delivery networks [CG09]. In all those cases a realistic simulation need to be run for a long simulated time. For the simulation approach to be affordable a storage device model is needed with a balance between accuracy an performance.

An alternate approach are given by black-box models where no knowledge of the storage device is included. The simplest black-box models are table-based models [And01], although more elaborate models exist [MWS+07, WAA+04].

Using a black-box approach, the behavior of the storage device can be modeled using a sequence of random variables T_i, which models the time required to service the i-th request. That sequence of random variables is a stochastic process. The goal of a simulation model is then to generate values fitting that stochastic process. To obtain such a simulation model experimental data must be obtained and analyzed to find the distribution behind.

However, the real reason for finding the distribution behind experimental data is the ability to generate random values accordingly with that distribution. A second approach is the generation of such values either from a experimental data histogram representation or from a experimental data cumulative distribution function representation.

3.2 Method

The approach proposed on this thesis starts with a sequence of disk requests (from a synthetic workload or real traces repositories) and ends up with a variate generator capable of producing several instances of simulated service time traces. The method has several steps as it is shown in Figure 3.1:

1.- Obtaining sequences of I/O requests

2.- Measuring service time

3.- Building the variate generator

4.- Producing simulation results

The next paragraphs explain every step in detail.

The first data set (step 1) is a sequence of I/O requests used to measure service time of a disk. That workload may come from two different sources:

- Real I/O traces from a specific system. This approach has the advantage of being very realistic. However, from a stochastic simulation point of view, they do not allow running different realizations of a experiment with different input.
- Synthetic workloads modeling a set of I/O traces realizations. This approach may be seen as not so realistic, but it will easily allow running different realizations of an experiment with different input.

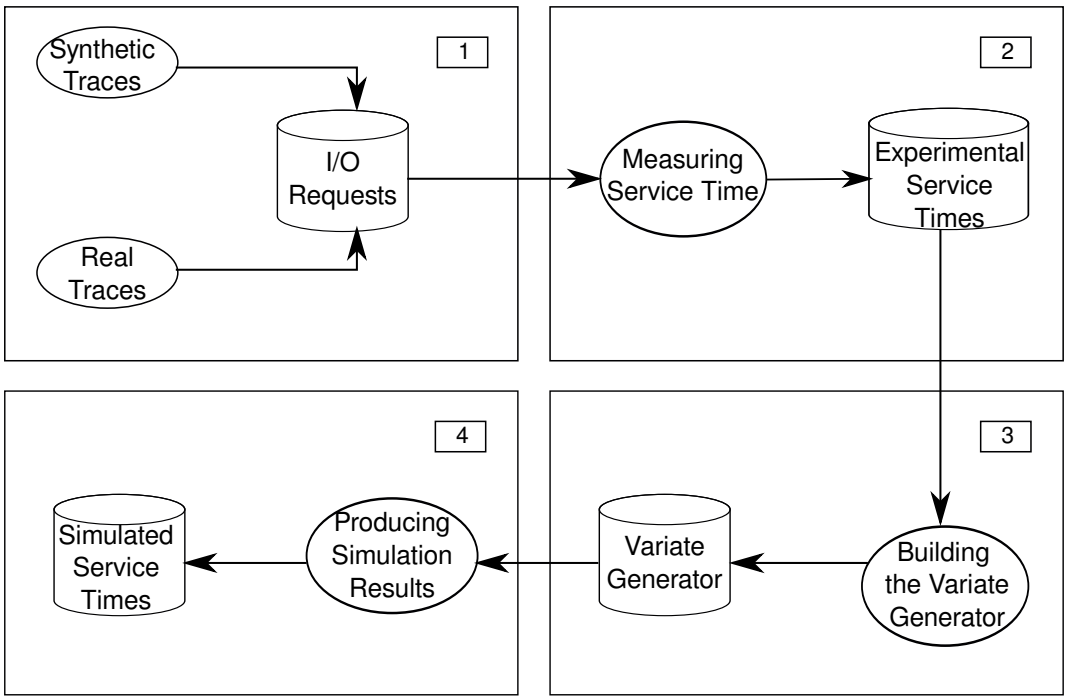


Figure 3.1: Steps in black box modeling of storage devices.

The selection of the source for the I/O requests depends on the purpose of the simulator to be produced. Our method allows to work with real I/O traces as well as with synthetic workloads as soon as a common representation is used. It is important to remark that the simulation module produced with our method is dependent on the I/O requests data sets. If we use I/O requests data sets coming from a Web server, the generated simulation module will be accurate to simulate disk service time under those load conditions and not under the load given by a database server.

Next step (step 2) is obtaining sets of experimental traces with measurements of disk service time. We perform different realizations of the measurement procedure. In each realization we send requests to the disk and we measure the service time for each individual requests, recording all the information in an experimental traces repository. Section 3.4 describes the details of this procedure.

After experimental measurement, obtained data must be analyzed to build a variate generator (step 3). Two approaches may be used here: building an analytical or an experimental variate generator. The analytical approach tries to find the distribution behind datasets. The experimental approach builds a variate generator from the histogram or the cumulative distribution function representations and can be easily automated. In this chapter we have selected the analytical approach.

Finally, the variate generator is included in a simulation module (step 4) to produce simulation results. After several realizations of the simulation are run with the initial workload used in step 2, simulation results are compared with experimental data to determine accuracy. Chapter 5 describes this step in detail.

3.3 Obtaining sequences of I/O requests

As we have already mentioned, we took the approach of using both synthetic and experimental traces. When using synthetic traces, we chose the SPC-1 benchmark [Cou06] to generate traces that are a common representation of other traces, and try to provide a wide range of characteristics from them. By using synthetic traces, we construct more general models, and although less accurate, usable by more kind of traces. On the other hand, when using real traces, we construct more specific models, mostly usable by traces that also have specific characteristics, detectable by those specific models. Moreover, specific models are usually more accurate.

3.3.1 Synthetic Workloads

To generate synthetic traces, we built a synthetic workload generator. Among different available options we chose the SPC Benchmark v1.10.1 [Cou06] workload defined by the Storage Performance Council [spc11].

The SPC-1 benchmark defines three Application Storage Units (ASU) each one representing a logical interface between a data repository and the host based programs. Among the different ASU to logical volume mappings defined by SPC-1 we have selected N-1 mapping in which the three ASUs are mapped to a single logical volume as our goal is to provide a model for a single disk.

The combination of the three ASUs defined in the specification (data store, user store and log) represent a typical usage of a storage system. Each ASU is composed of several I/O streams (4 streams for ASU1, 3 streams for ASU2 and 1 stream for ASU3) whose parameters are presented in Tables 3.1, 3.2 and 3.3.

	Stream			
Parameter	**1**	**2**	**3**	**4**
Intensity multiplier	0.035	0.281	0.070	0.210
Read fraction	0.5	0.5	1.0	0.5
Address	Uniform	RWalk	Inc	RWalk
size	8	8	Mixed	8

Table 3.1: ASU1 main parameters

	Stream		
Parameter	**5**	**6**	**7**
Intensity multiplier	0.018	0.070	0.035
Read fraction	0.3	0.3	1.0
Address	Uniform	RWalk	Inc
size	8	8	Mixed

Table 3.2: ASU2 main parameters

Parameter	Stream 8
Intensity multiplier	0.281
Read fraction	0.0
Address	Inc
size	Mixed

Table 3.3: ASU3 main parameters

For each I/O stream, the SPC-1 defines several parameters. The main parameters are an intensity multiplier specifying the contribution of that stream to the whole workload, a read fraction specifying the amount of read operations (rest are write operations), the start address of each operation and the size of that request. Addresses are aligned to a limit of 8 blocks with a block size of 512 bytes. Sizes of blocks are expressed in number of 512-bytes blocks.

As previously mentioned, the access pattern of each stream is highly dependent on the address and the transfer size. The address may be given by an uniform distribution (Uniform), a hierarchical reuse random walk [McN00, DF05] (RWalk), or an incremental access pattern (Inc). The size of the request is the amount of 512-bytes blocks to be read or written. Usually this parameter is fixed to 8 blocks. However, streams 2, 7 and 8 use a mixed model where request sizes vary from 8 blocks to 128 blocks.

The SPC-1 benchmark also defines Business Scaling Units (BSUs) which represent users accessing to the three ASUs. Each BSU is composed of the 8 streams previously described, and altogether provide a load of 50 operations per second. So the number of BSUs is directly proportional to the load of the disk, being 50 the coefficient of proportionality.

We have implemented a workload generator for the The SPC-1 benchmark. The goal of the workload generator is to get measurements for a high number of requests so that we are able to build a random variate generator for the disk. To achieve this goal the length of the experiments must be enough so that accurate estimations may be obtained. The usage of such lengthy experiments imposes an additional requirement on the underlying random number generator used, if correlation and autocorrelation are to be avoided. To guarantee that different random sources are based on non-overlapping random number generators we use the Mersenne-Twister [MN98] random number generator, which has a period equal to $2^{19937} - 1$.

In order to validate our implementation of the SPC-1 benchmark, we picked among 8 streams, just the first three, because their start addresses follow, respectively, the Uniform, RWalk and Inc distributions. The rest of the streams in the same and in other ASUs implement one of the three distributions, but with different parameters. We validated the chosen streams against the ones from an open-source implementation of the SPC-1 workload [DF05].

For stream 3, we also compared our Mixed size distribution implementation with the Mixed distribution from the open-source implementation.

As the random number generators for ours and the open-source implementation are

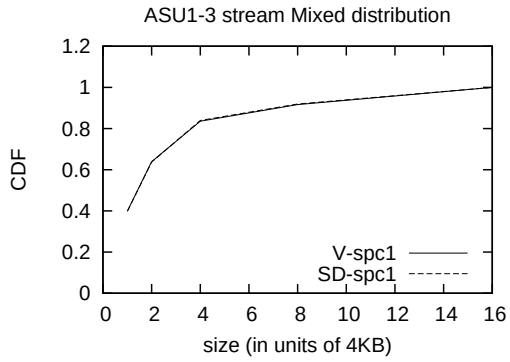


Figure 3.2: CDF results of running ours (V-spc1) and the open-source (SD-spc1) implementation of the SPC-1 benchmark for the Mixed size distribution

not the same, just using an equal seed may not cover the same addresable areas for both implementations. So we decided to force them to use equal inital addresses. From that moment on, we let both implementations take their own course.

Figure 3.2 shows the CDF (Cummulative Distribution Function) results of running ours (V-spc1) and the open-source (SD-spc1) implementation of the SPC-1 benchmark for the Mixed size distribution. The sample was of 100,000 requests. Note that they match quite well.

Figure 3.3 shows the CDF results of both implementations of the spc-1 benchmark for the Uniform (ASU1-1), RWalk (ASU1-2), and Inc (ASU1-3) distribution. Their samples were of 10 million requests for the Uniform and RWalk distributions, and of 100,000 requests for the Inc distribution. As we wanted, the initial addresses were equal. That made that also the covered area was the same and the CDFs overlapped. As a consecuence of that, the figure shows quite well the good fit of the distributions.

3.3.2 Real traces

In order to construct our more specific models, we use several different block-level representative traces, common in data-intensive I/O systems. Financial [uma11] is the I/O core of an OLTP application gathered at a huge financial organization. It performs about 5 million requests over 24 disks. Cello99 [Cel11] is a shared compute/mail server from HP Labs. It performs about 6 million requests over 25 disks.

Researchers have investigated several I/O intensive parallel scientific applications, such as MadBench2, S3D, and BTIO, to mention a few.

MadBench2 is a benchmark derived from a cosmology application that analyzes Cosmic Microwave Background data sets. MadBench2 spends approximately 81% of its total run time within MPI-IO read or write calls. Its most unusual characteristic is that it spends a significant portion of its time overwriting data. The out-of-core algorithm for MadBench2 resulted in each process alternating between read and write several times within the same file (7 per process). MadBench2 offers little opportunities for MPI-IO optimization or tun-

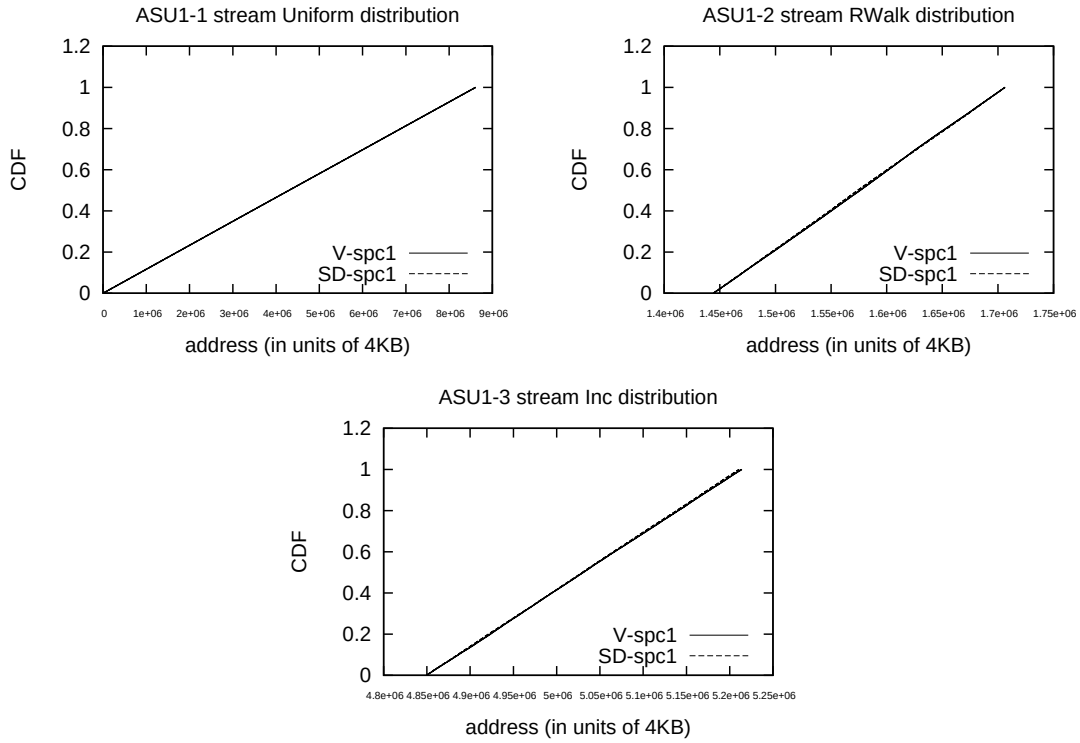


Figure 3.3: CDF results of both implementations of the spc-1 benchmark for the Uniform (ASU1-1), RWalk (ASU1-2), and Inc (ASU1-3) start address distribution

ing, since all data is accessed contiguously by using named datatypes and independent access [BIC10].

S3D is a parallel turbulent combustion application using a direct numerical simulation solver developed at Sandia National Laboratories. A checkpoint is performed at regular intervals, and its data consists primarily of the solved variables in 8-byte three-dimensional arrays, corresponding to the values at the three-dimensional Cartesian mesh points. S3D uses the pnetCDF collective interface, but through the use of MPI-IO hints we were able to evaluate both collective and independent I/O. Application performs one million 8-byte writes using independent I/O mode. On Blue Gene, this workload is a recipe for disaster due to I/O forwarding latency and the lack of write caching at compute nodes [BIC10].

NASA's BTIO benchmark solves the Block-Tridiagonal (BT) problem, which employs a complex domain decomposition across a square number of compute nodes. Each compute node is responsible for multiple Cartesian subsets of the entire data set. The execution alternates computation and I/O phases. Initially, all compute nodes collectively open a file and declare views on the relevant file regions. After each five computing steps the compute nodes write the solution to a file through a collective operation. At the end, the resulting file is collectively read and the solution verified for correctness [BIC10].

We have also evaluated our method, using real traces of the previous cited high performance applications. In case of S3D, we count with traces obtained from Red Storm [s3d11]. This system is a Cray XT3+ class machine, which uses Lustre as file system. The block

size used was 1MB and the storage system counted with 320 magnetic disks, in which file blocks are mapped round-robin over all disk. In case of BTIO and MadBecnh, the traces were obtained on a cluster with 4 I/O nodes, which uses PVFS as parallel file system. The storage system counted with 4 magnetic disks, in which file blocks are mapped round-robin over all disks.

3.4 Measuring Service Time

Once a workload is available, the next step is to measure the response time for each request. It is noticeable to remark that the goal here is not to get an overall metric (such as mean response time), but to get a measurement for every request as a prior task to build a simulation model. A workload is a file which has as many lines as I/O requests. Each line contains information about a single request such as the address of the disk at which is targeted, its size, the type of operation and the timestamp when the request should launch. So, a performance evaluator takes as input a previously obtained workload. An output file with the response times is generated.

To the best of our knowledge, there is a program which might achieve quite well this objetive. It is called *dxreplay* and belongs to a package of programs that comes from the tool *DIXtrac* [SG00, SGLG02, BS02]. Having a certain workload, *dxreplay* performs response time measurements on SCSI disks to compare them with the measurements taken from Disksim. It is for validating the correct extraction of the parameters that conform the detailed model of the real disk in DiskSim. It is composed of a main stream, which creates three instances of three threads: *lbnreader*, *issuer*, and *collector*. *lbnreader* reads requests from an workload input file, *issuer* launches the previously read requests to the disk at its specific time stamp, and *collector* waits for the launched requests to be finished. The three threads execute by turns and synchronize by using mutexes. However, *dxreplay* only works with SCSI disks. Therefore, we implemented *play*, our performance evaluator program that performs response time measurements on any kind of disk, and hence models any kind of disk, with any kind of interface.

Unlike *dxreplay*, *play* gets measurements by using standard POSIX calls and not SCSI commands. It is composed of a main stream which reads every request from the workload and launches it at the specified timestamp (see Algorithm 1). It also contains a signal handler which turns on when a specific request has finished and its response time can be reported directly to the output file, when making debugging tasks, or recorded on an in-memory data structure (see Algorithm 2).

Servicing multiple outstanding requests is a common functionality in nowadays disks. That is the reason behind our evaluator implementation uses asynchronous I/O. Every operation from the workload is issued at the specified start time and to roughly measure the response time. When a request is finished the service time is recorded on an in-memory data structure which is dumped on program termination. Both *play* and *dxreplay* generate output files containing response times, which can be represented by their CDFs as shown in Figure 3.4

To perform measurements on a given storage device to obtain the service time, we

Algorithm 1 play

1: $activate(signal\_handler, wake\_up\_when\_a\_request\_finishes)$
2: $loadRequestsInMemoryFromWorkloadFile()$
3: **for** $i \Leftarrow 1$ to $maxNumberOfRequests$ **do**
4: $recordStartTimeOfRequest(i)$
5: $lauchRequestToDiskAtItsTimeStamp(i)$
6: $waitForNextRequest(i)$
7: **end for**
8: $writeResponseTimesToOutputFile()$

Algorithm 2 signal_handler

1: $recordEndTimeOfRequest(finished\_request)$
2: $calculateResponseTime(finished\_request)$
3: $reportResponseTimeOnDataStructure(finished\_request)$

mounted the disk as an additional disk in a system having a primary disk, used for allocating the operating system, the evaluation tools and related files with the evaluation traces.

To avoid any impact in the evaluation procedure from the file system layer and the device driver and to ensure that every I/O request in the workload in physically sent to the evaluated disk, we use the GNU/LINUX Operating System and we define the disk as a raw character device by defining a character device (with mknod) and linking it to the disk that is going to be evaluate (with raw). In that way, the device can be accessed directly by using the **read** and **write** POSIX calls. Moreover, all the I/O operations are run over the address space of the evaluated process by means of DMA, making the addresses both in disk and memory be aligned to a 512 bytes boundary.

Possible noise generated by other running processes is taken to minimum, by running the evaluation tool with a minimum number of active processes (by using **init 1** mode).

As we will show in our evaluation, our performance evaluator program is validated

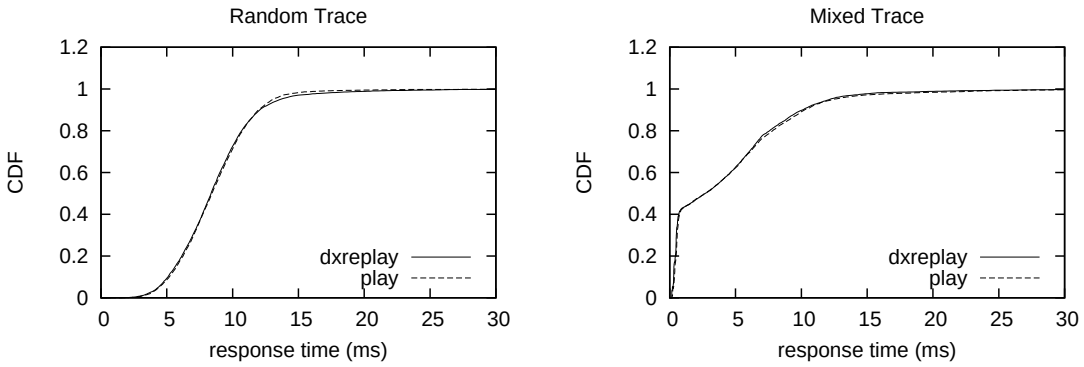


Figure 3.4: CDFs of response times from a Seagate Cheetah 10K.7 disk and two synthetic traces. Response times were generated by running both *play* and *dxreplay*.

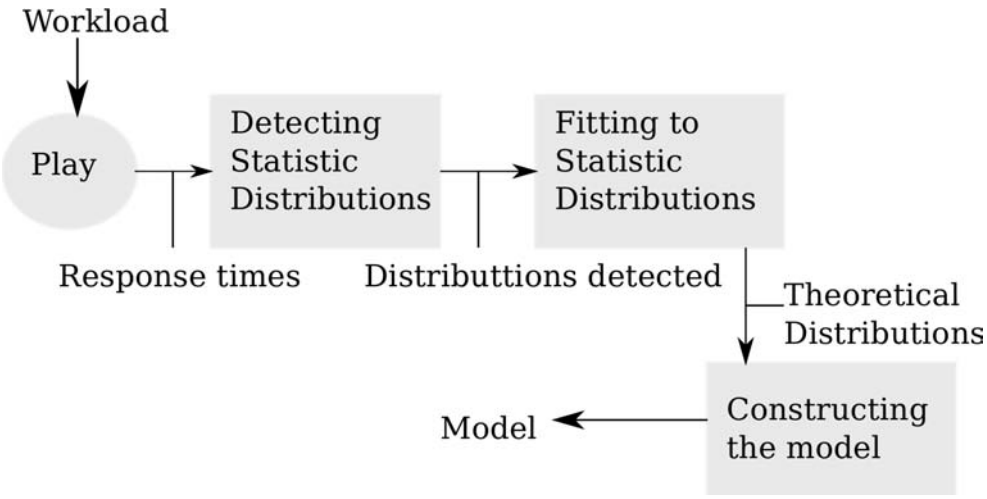


Figure 3.5: Steps in the process of Building the Variate Generator.

against *dxreplay*.

3.5 Building the Variate Generator

We now describe how to fit the previously obtained response times into one or several known distributions, to build a variate generator for such distributions. When building a variate generator, we start with a sample of response times and we end up with a variate generator capable of producing simulated response times. Our method has several steps as it is shown in Figure 3.5:

1.- Obtaining response times

2.- Detecting statistic distributions

3.- Fitting to statistic distributions

4.- Constructing the model

The next paragraphs explain every step in detail.

Our first data set (step 1) is a sample of response times obtained from replaying a trace on a real disk. That sample comes from running *play*, our response time measurement program for a specific workload. As previously said, the workload can be real or synthetic.

Next step (step 2) is detecting statistic distributions from the previously obtained sample of response times. The sample may present several different distributions that cover different ranges of the response time domain. Each distribution can be as a result of different causes that generate response times from one distribution or the others. As causes we can say that a specific request may be serviced from the platters of the disk or from its buffers, may have to wait for the previous requests to be serviced, etc.

After detecting possible distributions, the identified distributions must be fitted to some known distributions (step 3). Each individual distribution may be a single distribution or a mixture. Either single or mixture of distributions, we use the R statistical analysis environment [r-c11] to fit the samples to theoretical distributions. When fitting single distributions, we apply a method, and when fitting mixtures, we apply another method.

Finally, the fitted distributions are used to construct the model (step 4). Here, we propose two different approaches: One based on real traces and another based on synthetic traces. The one based on real traces consists on constructing individual models for several real traces. When response times are predicted from a specific trace, one of the previously modeled traces is chosen, and response times are generated from it. On the other hand, the one based on synthetic traces consists on constructing just one model based on a common synthetic trace. When response times are predicted from a specific trace, the model is able to partially adapt to the trace to predict.

3.5.1 Detecting statistic distributions

In our method, detecting statistic distributions has several steps:

1.- Representing the sample of response times by an histogram

2.- Detecting possible distributions

3.- Splitting the sample of response times in the detected distributions

4.- Fitting the detected distributions to theoretical distributions. If distributions cannot be fitted, go again to step 1.

When detecting statistic distributions, first step is representing the previously obtained data by an histogram. The histogram gives an image of the distributions that conform the previously obtained data. Each specific distribution represents one or several behaviours of the disk, under certain characteristics. For example, one of the distributions may represent response times generated by data hosted in disk caches. Another distributions may represent response times from data not hosted in disk caches. Other distributions may illustrate response times from workloads where the disk is idle for some specific periods. Some distributions may represent response times from very bursty workloads.

When detecting distributions, sometimes two different distributions are quite separated in the range of their domains, and it is easy to split them. Other times, they may be very close, and it is not so easy to know where to split. In those cases, what we do is splitting by where we consider, the value of joint. Then, we go to the next step, that is fitting the detected individual samples to theoretical distributions. If they can be easily fitted and the goodness-of-fit is good, we consider that the sample components are identified. Otherwise, we represent again the initial distributions by an histogram and try to detect the sample components and the value of joint.

Figure 3.6 represents the histogram's shape of a Seagate Cheetah 10K.7 disk under a Financial trace [uma11]. For simplicity, the cache was deactivated. Two distributions

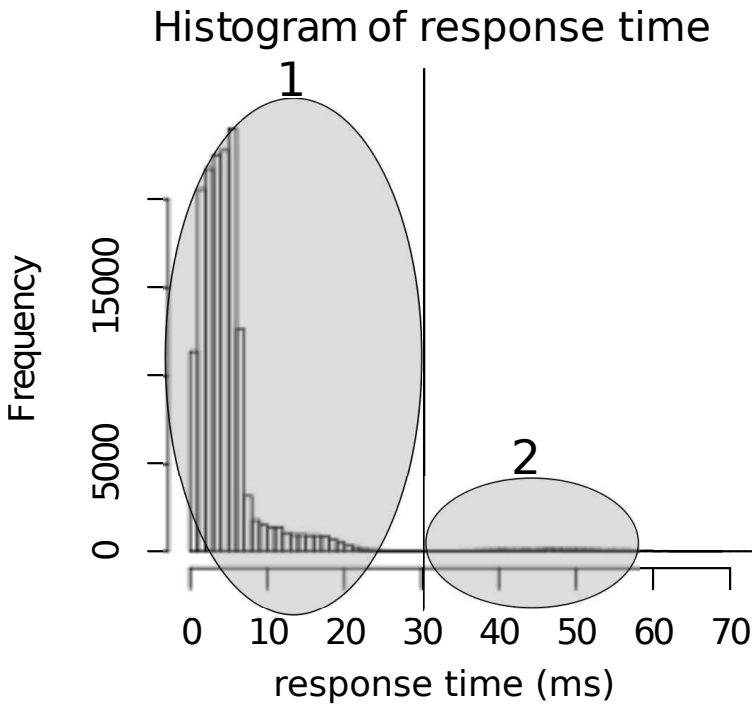


Figure 3.6: Distribution parts that represent service times from a Seagate Cheetah 10K.7 disk, under a Financial trace. Clearly, two distribution parts are identified.

are identified in the histogram: Distribution 1, which represents response times generated by the normal activity of the disk, and Distribution 2, which represents response times generated when the disk was previously idle, for some specific period of time. In this case, sample components were not close, and the value of joint was easily identified. As a value of joint we chose 30 ms. After fitting the sample components, and their goodness-of-fit, we considered we made a good choice, and the algorithm for detection was not repeated.

Once the distribution parts have been identified, next step is fitting them to some known distributions.

3.5.2 Fitting to statistic distributions

In our method, fitting to statistic distributions has several steps:

1.- Representing the sample component by an histogram.

2.- Determining if the sample component is a mixture of distributions.

3.- If the sample component is not a mixture, fitting it to a single theoretical distribution

4.- If the sample component is a mixture, fitting it to a theoretical mixture of distributions.

In order to fit a set of data to an unknown distribution, first step is inspecting its specific histogram, to determine if the experimental data presents a mixture of distributions. If a mixture is not identified, the *fitdistr* function from the R statistical analysis environment [r-c11] is executed. The arguments for this function are the experimental data, and the distribution to fit to. *fitdistr* is executed once per possible distribution (normal, lognormal, weibull, gamma, exponential), and returns the parameter estimates for such distribution. With the parameter estimates, the λ^2 discrepancy statistic [PJ90] quantifies the goodness-of-fit to all possible distributions and the distribution with the best fit is chosen.

If a mixture of distributions is recognized, the *mixdistr* package from the R statistical analysis environment [r-c11] is used. To fit a mixture, the first step is grouping the experimental data. For this, the *mixgroup* function is executed. The function groups the experimental data in the form of numbers of observations over sucesive intervals. The arguments are the experimental data and the number of intervals. In our case, we find the number of intervals by executing the function *binning* which automatically computes them. Next step is providing additional information about the shape and parameters of the experimental data. We do this by inspecting the histogram of the experimental data. We determine the starting values for the means and sigmas and make the proportions equal. Then, we estimate the parameters of the mixture, by executing the *mix* function. The arguments for this function, are the previously grouped data, the estimated parameters, the components of the mixtures (normal, log-normal, exponential, gamma, weibull, binomial, negative binomial, and poisson), the constraints, and the method/algorithms to employ. We usually consider a constant coefficent of variation as a starting constraint and use the combination of the EM algorithm [Man92] and Newton-type method. When using the EM procedure, the number of steps needs to be provided. For us, 3 were more than adequate.

In the example of Seagate Cheetah 10K.7 disk, under the Financial trace [uma11], both partial detected distributions are mixtures. Specifically, both distributions are mixtures of two normal distributions which parameters are presented in Table 3.4. Their shapes are also shown in Figure 3.7.

	type	π	μ	σ
Distribution 1	Normal	0.89	3.73	1.96
	Normal	0.11	10.82	5.71
Distribution 2	Normal	0.92	47.04	6.38
	Normal	0.08	29.42	3.99

Table 3.4: Parameters of the probabilistic distributions that model the service times obtained from a Seagate Cheetah 10K.7 disk, under a Financial trace.

After fitting the distributions to the experimental data, we see how well we have chosen the parameters by plotting the histogram for the experimental data with the estimated distributions. Figure 3.7 gives an example of such visualization. We also employ another visual techniques to judge the goodness-of-fit of a particular distribution to our experimental data, like the Q-Q Plot (Figure 3.8 - left). In the Q-Q plot, the more the blue line approximates to the diagonal, the best the goodness-of-fit because the best the fit between

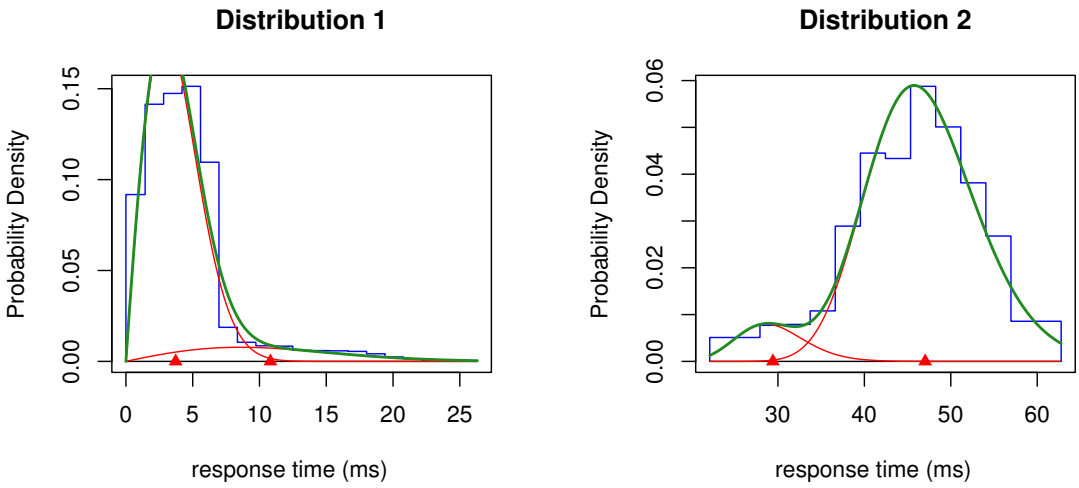


Figure 3.7: Comparison between two histograms, from experimetal data, and the probabilistic distributions they are fitted to.

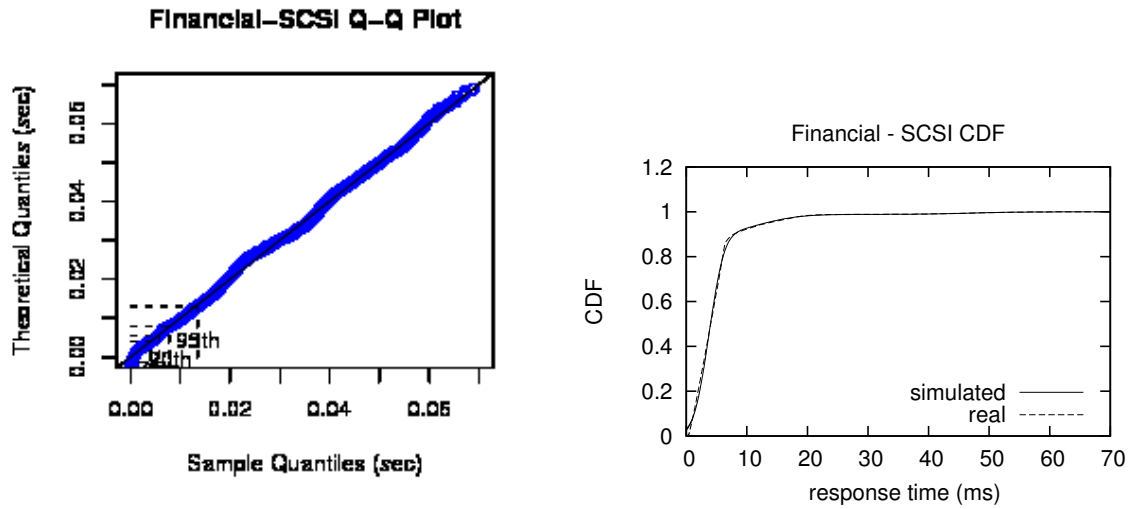


Figure 3.8: Two visual techniques to see how well simulated distributions fit real distributions. Q-Q plot compares quantiles of both distributions (left). CDFs' superimposion let visualize how well distributions fit (right).

the quantiles is. Among different numeric measurements, the one we used, for comparison reasons, was the demerit [RW94]. It is defined as the root mean square of the horizontal distances between the predicted and real response time cumulative distribution functions. We also represented the predicted and real CDFs to see how well they superimpose (Figure 3.8 - right).

Next step is building a simulation module by using the simulation environment OMNET++ [omn12]. The module simply hosts one or more random variate generators that generate simulated response times from previously fitted probability density functions.

To guarantee that different random sources are based on non-overlapping random number generators we use the Mersenne-Twister [MN98] random number generator, which has a period equal to $2^{19937} - 1$.

3.5.3 Constructing the model

In our method, constructing models has several steps:

1.- Choosing the type of traces in which the model is based.

2.- If traces are real, constructing a model for each of the identified and fitted distributions of those real traces

3.- If the trace is synthetic, constructing a model for the synthetic trace without taking into account the enqueing times

On the other hand, predicting response times from the previously constructed models has several steps:

1.- Choosing the type of traces in which the model is based.

2.- If traces are real, predicting from one of the modeled real traces, by identifying the one which has the most similar characteristics as the trace to predict.

3.- If the trace is synthetic, predicting from the synthetic modeled trace, and calculating the enqueuing times on the fly.

The next paragraphs explain every step in detail.

As we previously said, in our method we proposed two kinds of approaches, in order to construct simulation models for disk drives. The first one is based on several real traces with similar characteristics as the workloads to predict. The other one is based on a synthetic workload, which is supposed to cover a wide range of characteristics and simulate any kind of trace. So first step is choosing the approach which may be better adapted to the needs of the system to simulate or predict.

Next step is constructing the models, which depends on the previously chosen approach. If the chosen approach is based on several real traces, we have to construct a model for each of the real traces, on which the model is based. When constructing a model for a specific trace, we identify reasons to choose one or another identified and fitted distributions. In the next section, we will explain this in more detail. On the other hand, if the chosen approach is based on a synthetic trace, we have to construct only a model for that specific trace. When constructing the model, we do not take into account enqueuing times, which are one the most influential factors in the response times, for the models to be more versatile and general.

After constructing the models, we can use them to predict response times from them. As previously said, depending on the needs of the system to simulate or predict, we choose the first or the second approach. When choosing the first approach, one of the real traces in which it is based, has to be chosen. Once it is chosen, response times are simulated from it. In the course of the simulation, the characteristics of the input workload may change, and a new real trace may also have to be chosen. On the other hand, when choosing the second approach, response times are directly simulated from the synthetic trace. As we said, when modeling a synthetic trace, we do not include queuing times in the model, and leave it until simulation, making the model more versatile.

In this section, we propose two approaches to construct simulation models by using probabilistic distributions. The first approach is based on one or several real traces. The second approach, is based on a common synthetic trace, which represents a wide range of workload characteristics. Although the proposed approaches are different, when constructing the models, both have several steps in common. Those steps are described in the next section.

3.5.3.1 Constructing models

When constructing models, what we actually do, is figuring out the reasons for the previously detected, and fitted-to-theoretical-distributions samples. For reasons, we mean the workload characteristics that produce response times from a certain distribution and not from the others. Among several reasons we can find:

- Long inactivity periods. When the disk has been idle for a certain amount of time, the next request to be serviced may last longer than as usual.
- Queuing times. When a request to be serviced arrives when one or several previous requests have not finished yet, it must wait until the end of the previous ones, making its response time bigger.
- Sequentiality. When several requests are sequential, the difference in LBNs between them is little. On the contrary, when the difference in LBNs is not little, sequentiality may not be identified and response times are bigger.
- Caching effects. Disk drives have internal buffers. When their usage is activated, servicing requests from the disk drive buffers is faster than servicing requests directly from the platters.

So, once we have identified the reasons for generating response times from each specific fitted distribution, we construct a model in which, depending on the input data, we choose one theoretical distribution or other, to generate response times.

In the example of Seagate Cheetah 10K.7 disk, under the Financial trace [uma11], we identified two distributions and three causes for them. As we previously said, Distribution 1 is a mixture of two Normal distributions (see Table 3.4). Response times from part 1 of Distribution 1 are generated when requests must be serviced from the platters. Response

times from part 2 of Distribution 1 are generated when inactivity periods are longer than 1 second and when requests serviced from the platters, must wait until the end of the previous ones. Finally, response times from Distribution 2 are generated when inactivity periods are around half a second. So, for this specific model, we constructed an algorithm as shown in Algorithm 3:

Algorithm 3 Model for a Seagate Cheetah 10K.7 disk, under the Financial trace

```
1: if ((simTime()-ini_disk[actReq − 1]) > 520)&((simTime()-ini_disk[actReq − 1]) <
   700) then
2:    choose ⇐ bernoulli(0.92208)
3:    if choose = 1 then
4:       response_time ⇐ normal(47.04, 6.385)
5:    else
6:       response_time ⇐ normal(29.42, 3.993)
7:    end if
8: else
9:    if ((simTime()-ini_disk[actReq − 1]) > 1000)‖(simTime() < end_disk[actReq − 1])
      then
10:      response_time ⇐ normal(10.819, 5.706)
11:   else
12:      response_time ⇐ normal(3.727, 1.965)
13:   end if
14: end if
```

The generation of response times from Distribution 2 is described in lines 1 - 8. Every time the disk has been idle for more than 520 ms and less than 700 ms, the actual request, $actReq$, generates its response time from one of the parts of Distribution 2. We determine that idleness by doing a substraction between the time stamp, when $actReq$ arrives ($simTime()$), and the time stamp when the previous request arrives ($ini\_disk[actReq-1]$). Probability of choosing part 1 of Distribution 2 is higher (0.92208) than choosing part 2. We determine which part to use by using a bernoulli distribution (line 2).

The generation of response times from Distribution 1 is described in lines 9 - 14. Every time the disk has been idle for more than 1000 ms, the actual request, $actReq$, generates its response time from part 2 of Distribution 1 (line 10). Also, if $actReq$ arrives ($simTime()$), and the previous request $actReq-1$ has not been serviced yet, response times are generated from part 2 of Distribution 1 (line 10). Thus, we simulate that $actReq$ has been enqueued. In any other case, response times are generated from part 1 of Distribution 1 (line 12).

As we will show in our evaluation, activating the use of the internal buffers will generate response times from another distribution, whenever requests must be serviced from the drive buffers.

3.5.3.2 Constructing models based on real traces

When constructing models based on real traces, we take, for each real trace, the same steps described in the previous section. So, for each real trace we distinguish among several

reasons to generate response times from one fitted distribution or from the others, if there exist more than one distribution. If there exist only one distribution, all response times are generated from it.

Finally, what we actually have is a pool of models from several real traces of different characteristics. Predicting from this model involves choosing one of the several previously modeled traces. The criteria to select a trace are based on the mean request size and the mean queuing time of the trace to model. Sequentiality is also taken into account. On the basis of the mentioned criteria, a previously modeled trace is selected and used to predict response times from an input workload. Every now and then, the criteria are checked again, and if they have changed, another most similar previously modeled trace is chosen. For this approach to be accurate, there must be at least one previously modeled trace with similar characteristics as the trace to be predicted. Algorithm 4 shows how the prediction in this model works:

Algorithm 4 Model based on several real traces for a Seagate Cheetah 10K.7 disk

```
1: if (contLastReqs < lastReqs) then
2:    if (distr = 0) then
3:       response_time ⇐ generate_S3D()
4:    end if
5:    if (distr = 1) then
6:       response_time ⇐ generate_BTIO()
7:    end if
8:    if (distr = 2) then
9:       response_time ⇐ generate_MadBench()
10:   end if
11:   if (distr = 3) then
12:      response_time ⇐ generate_Financial()
13:   end if
14:   if (distr = 4) then
15:      response_time ⇐ generate_Cello99()
16:   end if
17:   contLastReqs + +
18:   addReqSizeToStatistics()
19:   addQueuingTimeToStatistics()
20:   addSequentialityToStatistics()
21: else
22:   calculateDistr()
23:   contLastReqs ⇐ 0
24: end if
```

Before the first request arrives, *contLastReqs* is initialized to 0. *lastReqs* is a constant which determines the maximum number of arrived requests, before checking if another distribution has to be selected. When a distribution has to be selected, the method *calculateDistr*() is executed (line 22). This method determines, on the basis of some criteria, which distribution is the most appropiate for the input trace to predict. According

to the selected distribution, which value is in variable *distr*, response times are generated from it (lines 2-16). Every time a response time must be generated for a specific request, the statistics for the input workload must be also updated. Those statistics include the mean request size (*addReqSizeToStatistics*(), line 18) , the mean queuing time (*addQueuingTimeToStatistics*(), line 19), and sequentiality
(*addSequentialityToStatistics*(), line 20).

3.5.3.3 Constructing models based on synthetic traces

When constructing models based on a synthetic trace, we also take, for the synthetic trace to be modeled, the same steps described in Section 3.5.3.1. For this construction, we also distinguish among several reasons to choose among one fitted distribution or the others, in case there exist more than one.

Probably the big difference with regard to constructing models based on real traces is that, besides it uses only one trace, queing times are not modeled in the extracted response times. We can do that by using a characteristic of *play*, our response time measurement program. As previously said, it lets the user extract response times by avoiding queuing times. It can be done by restricting the maximum number of pending I/O requests that can be enqueued to the disk to 1. Thus, queuing times are modeled online in the predictions. As queuing times are one of the most significant and distinguishing effects in the response times, if every input trace predicts them on its own, models can be more versatile and general. The purpose of this is finally to have a more versatile model by simulating the queuing times on the fly. In Algorithm 5 we show, for the previously constructed model of the Seagate Cheetah 10K.7 disk, under the Financial trace [uma11], how to calculate queuing times on the fly. Although the chosen trace is not synthetic, our goal here is to show the difference between generating queuing times from a previously modeled distribution and when generating them on the fly.

As previously said, the generation of response times from Distribution 2 is described in lines 1 - 8. Every time the disk has been idle for more than 520 ms and less than 700 ms, the actual request, *actReq*, generates its response time from one of the parts of Distribution 2. We determine that idleness by doing a substraction between the time stamp, when *actReq* arrives ($simTime()$), and the time stamp when the previous request arrives ($ini\_disk[actReq-1]$). Probability of choosing part 1 of Distribution 2 is higher (0.92208) than choosing part 2. We determine which part to use by using a bernoulli distribution (line 2).

The generation of response times from Distribution 1 is described in lines 9 - 14. Every time the disk has been idle for more than 1000 ms, the actual request, *actReq*, generates its response time from part 2 of Distribution 1 (line 10). In any other case, response times are generated from part 1 of Distribution 1 (line 12). If *actReq* arrives ($simTime()$), and the previous request $actReq-1$ has not been serviced yet, *actReq* is enqueued. To simulate that it has been enqueued, we calculate its response time, by adding the time that it stays in the queue ($end\_disk[actReq-1] - simTime()$ - line 16) up to its previously calculated service time ($response\_time$) .

Algorithm 5 Model for a Seagate Cheetah 10K.7 disk, under the Financial trace. Queuing times are calculated on the fly

1: **if** $((simTime()\text{-}ini\_disk[actReq-1]) > 0.52)\&((simTime()\text{-}ini\_disk[actReq-1]) < 0.70)$ **then**
2: $choose \Leftarrow bernoulli(0.92208)$
3: **if** $choose = 1$ **then**
4: $response\_time \Leftarrow normal(47.04, 6.385)$
5: **else**
6: $response\_time \Leftarrow normal(29.42, 3.993)$
7: **end if**
8: **else**
9: **if** $((simTime()\text{-}ini\_disk[actReq-1]) > 1)$ **then**
10: $response\_time \Leftarrow normal(10.819, 5.706)$
11: **else**
12: $response\_time \Leftarrow normal(3.727, 1.965)$
13: **end if**
14: **end if**
15: **if** $(simTime() < end\_disk[actReq-1])$ **then**
16: $response\_time \Leftarrow response\_time + (end\_disk[actReq-1] - simTime())$
17: **end if**

3.6 Summary

In this chapter, a method to construct a black box model for hard disk drives has been described. It is based on probability distributions and can be used both for synthetic and real traces. The method includes a measuring service time implementation which can be used in every disk, with any kind of interface.

Two approaches have been proposed for the construction of the models. The first approach, is based on multiple real traces. It may be accurate when the traces to predict have similar characteristics as one of the traces, for which the model has been constructed. The second approach is based on a single synthetic trace. This trace is supposed to cover a wide range of characteristics from other traces. This approach calculates queing times on the fly, on the prediction stage, which makes it more versatile.

As will be shown in the evaluation (Chapter 5), the method described on this thesis provides an accurate, and more general, alternative approach, to model service times from any device, with any kind of interface. This is because, unlike other approaches [DIX12], its measuring service time implementation uses standard POSIX calls. Also, as it constructs a black box model, many of the device characteristics do not have to be known, making it simpler, less detailed, and hence, ligther. In the evaluation will also be shown that the proposed method may be as accurate as another detailed models [dis08], and also provides a better performance, making it affordable in system wide simulations.

Unlike other black box models [And01, WAA+04, MWS+07, LH10], in which predictions are made by searching information of input data from workload, in big tables and regression trees, the proposed approach, predicts service times by choosing, at most, among

five different probability distributions, which makes it faster when predicting.

The proposed method and its experimental results have been published in:

- *A Black Box Model for Storage Devices based on Probability Distributions.* Laura Prada, Alejandro Calderón, Javier García, J. Daniel Garcia, Jesus Carretero. 10th IEEE International Symposium on Parallel and Distributed Processing with Applications. July, 2012.
- *Using black-box modeling techniques for modern disk drives service time simulation.* Jose Daniel Garcia, Laura Prada, Javier Fernandez, Jesus Carretero, Alberto Nunez. The 41th Annual Simulation Symposium (ANSS'08). April, 2008.
- *Modelado estocastico de las operaciones de entrada/salida sobre un disco.* Laura Prada, J. Daniel Garcia, Alberto Nuñez, Javier Fernandez, Jesus Carretero, Ramon J. Flores. II Congreso Español de Informática (CEDI 2007). XVIII Jornadas de Paralelismo. Zaragoza, España. September, 2007.

Chapter 4

Energy-aware Architectures

4.1 Introduction

In a computer system, disk drives are one of the most power consuming elements. This is mainly due to their mechanical nature. Disk drives usually have several platters in which data are stored. To access those data, several heads, moved by an arm, are used.

Energy consumed by disk drives affects both laptop/desktop environments, and data center or server environments. In the case of laptop/desktop environments, energy consumed by disk drives reduces the amount of power available in batteries or power supplies. In the case of data centers or server environments, energy consumption of disk drives increases, mostly, expenses in electricity bills, and also CO_2 dissipation. That is mainly because in data centers and server environments, a lot of disk drives are used in order to improve performance, by servicing I/O requests in a parallel fashion. Both in laptop/desktop environments and data center or server environments, the techniques to save power are usually the same two: The ones that apply power saving techniques to current disk drives, and the ones that apply power saving techniques to disks of several speeds [CPB03, GSKF03].

With an aim toward saving power, current disk drives have several states of energy: *active*, *idle* and *standby*. A disk is in active state when it is reading or writing data. When a disk is not doing anything, but is still spinning, its state is idle. When a disk is not doing anything and its platters do not spin either, its state is standby. The highest power consumption occurs at the active state. Idle state consumes less power than the active state, and standby state consumes much less power than the previous two states. The disk spins down to the standby state when the disk has been in the idle state for a certain period and it is predicted that the state change is worth it. When an I/O request needs to be serviced by a disk that is in the standby state, it has to spin up before anything, and then service the I/O request. Spinning up a disk drive takes several seconds and energy costs.

On the other hand, disk drives with several speeds, although not available in the market yet, are able to modelate their speed according to the load. The higher the load, the higher the disk speed. When the disk load becomes lighter than a certain disk throughput, the disk spins down to a lower speed mode; if the load is heavier than a certain threshold, the disk spins up to a higher speed. The main advantage of these disks is the fact that, as they move gradually from one speed to another, it does not take so much time and energy as it takes in traditional disks.

In this thesis, we focus on traditional disks. As previously said, in traditional disks, it takes some time and energy, to move between energy states. So, when moving from the *idle* state to the *standby* state, the disk should stay in the *standby* state enough time for the subsequent (when spinning the disk up) waste of time and energy to be worth it. In order to optimize the amount of energy saved, when the disk has remained in the *standby* state, several *spin-down* algorithms have been proposed [LKHA94, DKB95, HLS96]. Those algorithms aim to decide when is the right moment to move from the *idle* state to the *standby* state, for the disk to stay in the *standby* state as long as possible, and to save energy. They learn from previous decisions taken, and how those decisions have worked. Spin down periods or idle times are broken when I/O requests, either reads or writes, come to the disk drive. When *spin-down* algorithms do things properly, they can predict when I/O requests come to the disk drive, and they act accordingly. However, *spin-down* algorithms cannot spin disks down frequently when idle times are not big enough, as it happens in HPC applications.

In order to prolong *standby* times in HPC applications, we propose to use supporting SSDs for the disk drives. That way, I/O requests can be redirected to the SSDs, thus achieving long *standby* times. Also, they can help to save power when idle times are not big enough. SSDs are light, silent, and consume less power than disk drives, because they have no mechanical parts. They have just two states of power: *active* and *idle*. A SSD is in active state when it is reading or writing data. When the SSD is not doing anything its state is *idle*. So, when redirecting I/O requests to supporting SSDs, disk drives may stay in the *standby* state for longer, thus, saving energy.

Saving energy by spinning disks down entails a trade-off. Every time a disk spins down and spins up, the heads and the spindle motor wear. That is why, manufacturers specify a maximum number of start/stop cycles that a disk drive can stand without producing errors, and having to be replaced. For desktop drives, start/stop cycles are around 50,000 and for laptop drives, around 300,000.

Also, redirecting most of the I/O requests to supporting SSDs means writing them with a lot of data, and also means a trade-off. SSDs can be written a limited number of times. As to rewrite a specific block, it must be erased first, what is really limited is the number of times that a block can be erased. When the number of maximum erasures is reached, the SSD's blocks become unerasable. There are two kind of NAND flash SSDs: SLC (single level cell) and MLC (multi-level cell). SLC provide only one bit per cell, while MLC provide several bits per cell, increasing the likelihood for errors. This increased likelihood for error also limites the number of times that a specific block can be written, which is around 10,000 times in MLC flash and 100,000 in SLC flash.

So, in order to save energy by spinning disks down, and using supporting SSDs, it is

important to take into account the limited number of start/stop cycles in disk drives, and also, the limited number of times that SSDs can be written.

4.2 Proposed Solution

Our work addresses the problem of finding a power saving-aware architecture for large parallel computing environments such as clusters or supercomputers. Our main goal is to provide a SSD-based power saving-aware architecture that can extend *idle* times in HPC applications. Figure 4.1 describes a common I/O architecture for parallel environments. A first approximation is to put compute nodes close to a local storage system. Current trend targets to eliminate local disks in order to reduce space, and to save energy. In this diskless solutions, compute nodes forward file access requests to I/O nodes, which aggregate and forward data to the permanent storage subsystem [BIC10]. Storage nodes can take advantage of SSD devices in the following way: independent disks can have an associated SSD device (as shown in Figure 4.1.a), or an array of independent disks (as shown in Figure 4.1.b). In both cases, a set of HDD and SSD creates a Storage Element (SE). SEs are accessed as a single physical disk by the applications. The Figure shows how hardware components map in the actual I/O subsystems. Compute nodes are connected through a fast-interconnect network to the I/O nodes. I/O nodes access to the storage system on behalf the compute nodes. Current trend targets to eliminate local storage to reduce compute nodes energy consumption. Storage nodes provide two approaches: Independent disks-based storage architectures (a), where each HDD device has a corresponding SSD device and RAID-based storage architectures (b), where a single SSD deals with an array of independent disks. In this thesis, we describe specific solutions for Independent disks-based storage architectures (a).

4.2.1 Overview

A parallel I/O subsystem based on multiple I/O nodes allows data transfer in parallel between compute nodes and I/O nodes. This architecture presents the following advantages: first, it can efficiently handle the small, fragmented requests produced by parallel programs. Second, by adding I/O nodes or disks, it can scale in bandwidth and capacity to keep up with increases in the number and speed of compute nodes. Finally, the parallel subsystem provides load distribution by scattering I/O operations across multiple nodes. It also can provide reliability when a system element is down or being replaced [BIC10].

In this thesis, we propose the SSD-based Power-aware Storage System (SSD-PASS) architecture, shown in Figure 4.2. It has three layers: compute, I/O, and storage nodes. Compute nodes execute parallel applications. Applications access I/O through libraries which invoke virtual file system interfaces, and they in turn invoke parallel file system interfaces to access I/O nodes through interconnection networks. SSD-PASS is deployed on storage nodes, where data are finally stored in magnetic disks. SSD-PASS has three main modules: Write-buffering, Prefetching, and Indirection map modules.

SSD-PASS lies on a hybrid architecture, in which each I/O node consists of a SSD

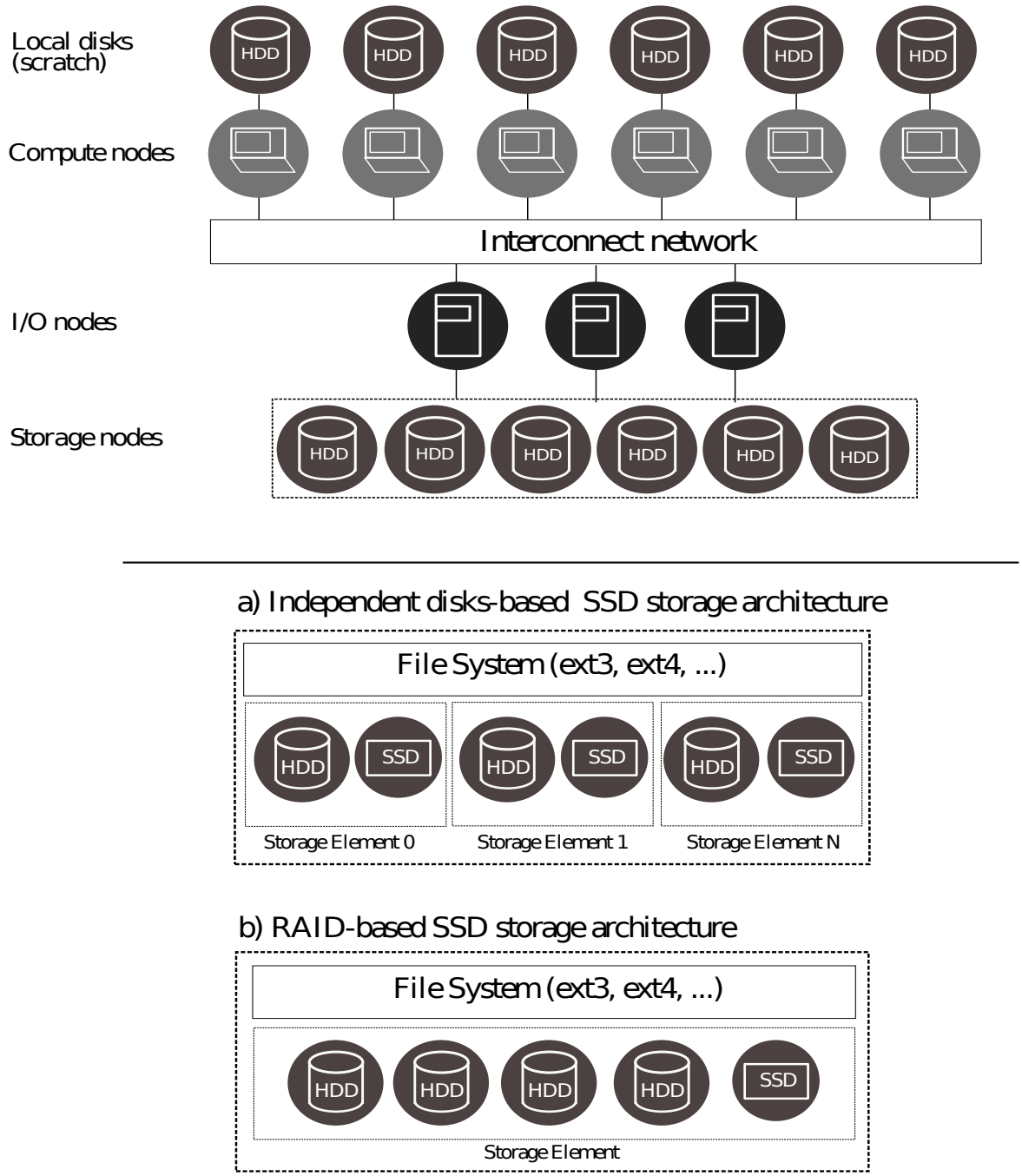


Figure 4.1: Typical supercomputer I/O architecture.

and a conventional magnetic disk device, as shown in Figure 4.2. Our general procedure aims to use the SSD device as a block cache for a specific file system on I/O nodes. I/O operations are transparent for both SSD devices and the parallel file system.

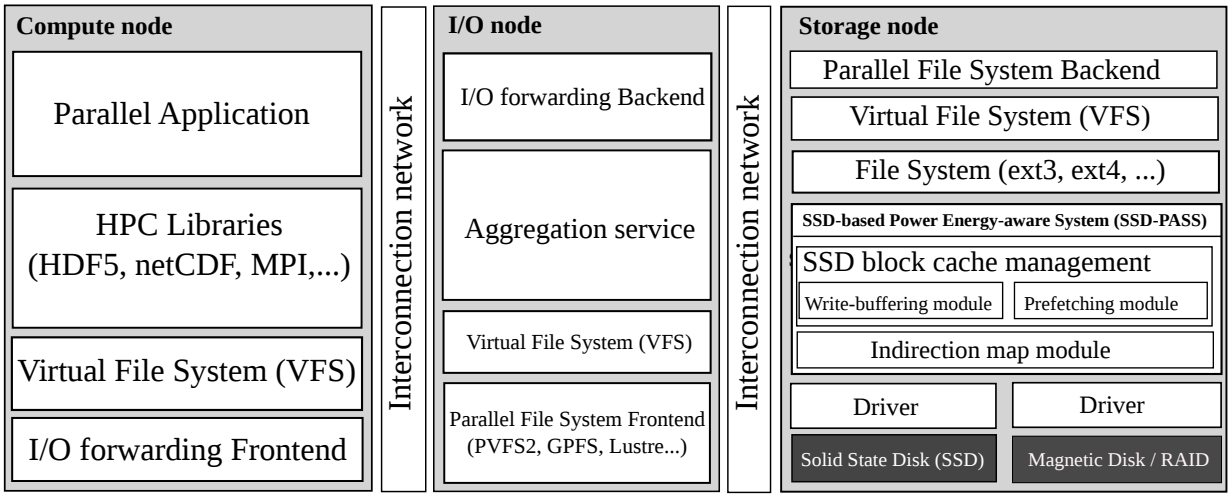


Figure 4.2: SSD-PASS architecture multi-level stack. The figure depicts the software architecture of all storage nodes.

4.2.2 Data accesses

Our solution takes advantage of an indirection map, which permits blocks to be allocated in different devices, mapping SSD and disk logical block addresses (LBAs). Every read and write request is sent to the corresponding device after mapping the indirection map, by using the actual physical location. Data consistency is enforced in the system by not allowing more than one copy of a file block at the SSD device. Our approach is based on a one copy consistency model. Either if the data is in one device or in both, there is a need to have the most accurate information in the remapping of the block addresses.

Applications access the SSD-based architecture through the POSIX [hs95] interface, common in HPC applications [GSKF03] and parallel libraries such as MPI [Mes95]. Additionally, our storage system takes advantage of the Virtual File System (as shown in Figure 4.2), which virtualizes file I/O operations. Subsequently, in order to use our architecture, both application and file systems do not need to be modified.

4.3 Power saving-aware write-buffering policy

In this section, the policies included in the previously mentioned write-buffering module, from the SSD-PASS architecture, are described.

The objective of a conventional write-buffering technique is to improve file access performance by saving file data requests in the memory buffer. Servicing requests from the memory buffer is faster than servicing them directly from disk drives. However, in this work we propose a power-saving aware write-buffering technique, which focuses on keeping the unwritten blocks in the SSD devices for as long as they have space enough. This way, periods when disk drives can stay in the *standby* mode, are dependant on SSDs sizes. The bigger the SSDs sizes, the longer the idle periods.

Memory buffers have two main problems when they are used for power aware goals. First, they are volatile, which means that when a power cut takes place, cached data are lost. Second, in many operating systems there are daemons which make sure that cached disk data are written to disk every now and then. This prompts the fact that long-idle intervals may be broken into not sufficiently long periods to spin the disks down, and save energy. Also, their sizes are not usually more than several GigaBytes. SSDs devices are non-volatile and their sizes range from 2 GB to 2 TB. Second, they can take advantage of shorter service demands that come from the not extreme head displacements at disk devices. As was previously said, SSDs do not have mechanical parts, and neither, head displacements.

The write buffering policy is enforced at the SSD and consist of (as shown in Figure 4.3):

1.- Redirecting writes to the SSDs as long as they have enough space,

2.- Spinning the disks up, and

3.- Flushing the SSDs contents back to disks when the SSDs are full.

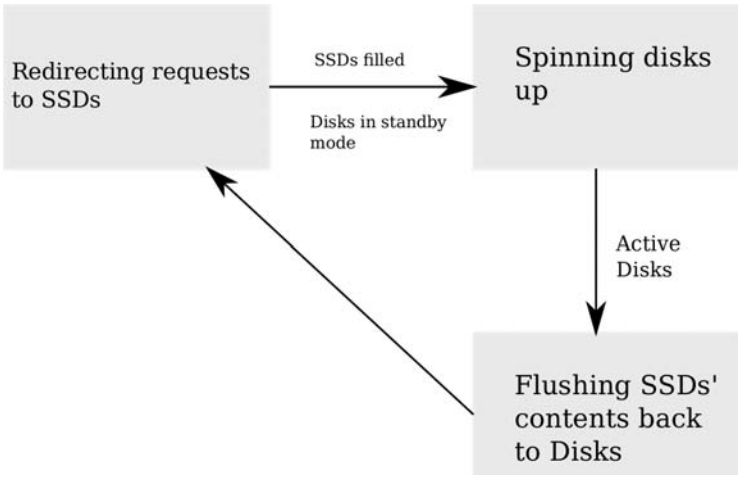


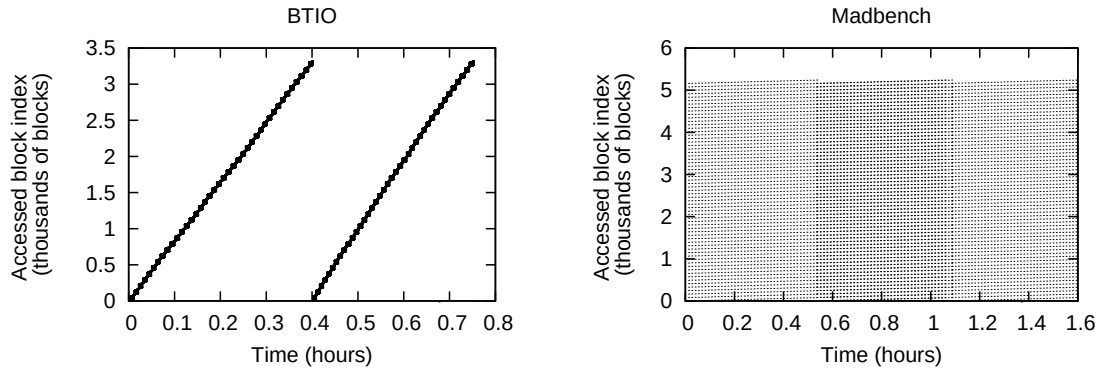
Figure 4.3: Steps in the write buffering policy.

When redirecting writes to the SSDs, we do it in a log-fashion manner. We locate the current request's contents after the previous ones. Thus, we take advantage of the sequential write performance of SSDs. Also, if the requests are sequential to some degree, when flushing the SSDs contents, reordering those contents becomes easier.

After being in the *standby* state for a while, disk drives must spin up to be ready to receive the written data from SSDs. As was previously said, spinning disk drives up takes time (in the order of several seconds), and consumes some energy. To avoid applications execution to waste time in spinning disks up, disks can be spinned up in advance. When doing that, disk drives should be spinned up, before SSDs are filled, the same number of seconds that takes for the disks to spin up. Thus, when SSDs are filled, disk drives will be

ready to receive the SSDs written contents. Spinning disks up in advance may consume more energy, but it makes applications take less time to finish.

Flushing the SSDs contents back to disks when SSDs are full takes some time and waste of energy to consider. First step is reading the written contents from SSDs. Next, data are reordered in order to take less time, to write the contents back to the disks. After data are reordered, contents are written back to disks. The time for flushing, will mainly depend on the SSDs size. The bigger the SSDs, the more time the flushing will take.

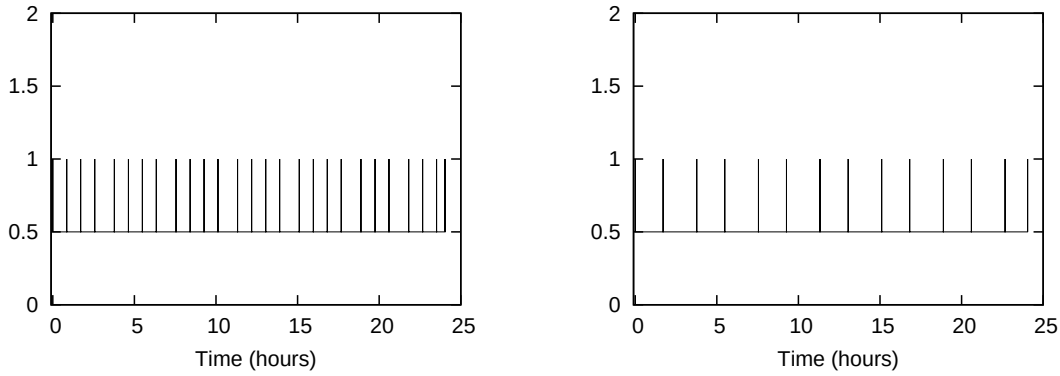


(a) BTIO (b) Madbench

Figure 4.4: HPC access pattens. The graphs represent two common access patterns which are typical in HPC applications.

Some applications exhibit repetitive behaviour. When I/O access patterns are repetitive, caching can avoid some data accesses to disks [BIC+09]. Those data can also stay for some time in the SSD devices for future reads. Thus, future disk accesses do not have to break idle periods, making them bigger. We write data in a log-structured SSD cache. This solution allows to take advantage of sequential write performance of SSDs. Moreover, after flushing SSDs contents, applications will start to overwrite old data. If they need to read previously cached data, as SSD sizes are big enough, applications will have enough time to read the data before overwriting them.

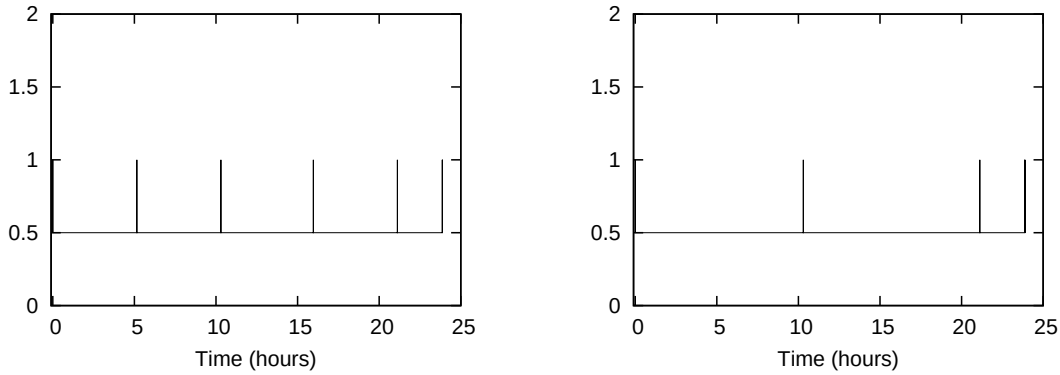
For example, as it is shown in Figure 4.4, two I/O intensive parallel scientific applications, such as BTIO [bti11] and MADBench2 [mad11], clearly exhibit repetitive behaviours. In the BTIO case, data are written until the first half part of the trace duration (0.4 hours). After that, the same written data blocks are read (see Figure 4.4(a)). So, in this specific case, when about 1.58 GB of data are written, and then read again, the usage of SSDs of at least 2 GB of size becomes adequate. This is because the application have space enough to write data, and then read them, before start overwriting them. Prolonging the duration of the application involves repeating the same patterns several times. When applications are writing data in SSDs, and they get filled, after moving the written contents from SSDs to disks, applications keep writing data by overwriting the first blocks of the SSDs, thus, overwriting already read data. Thus, whenever recently written data are read, all of them can be accessed from SSDs, without having to wake disks up. This way, disk drives are only waken up when SSDs are filled, and hence, contents need to be moved to disks. Figure



(a) BTIO (b) Madbench

Figure 4.5: Ups (active state - 1) and downs (standby state - 0.5) in the course of time, of a single disk drive, when the write buffering policy is applied. SSDs are 2 GB sized.

4.5(a) shows a time line of the effects in disk drives of the BTIO application. The figure shows the ups (active state - 1) and downs (standby state - 0.5) in the course of time, of a single disk drive, when the proposed write buffering policy is applied. In this case, the size of SSDs is 2 GB. All the shown ups are due to the fact that SSDs get filled, and contents must be written to disks. Figure 4.6(a) shows the ups and downs in a time line when SSDs are 4 GB sized. As can be seen, ups are less frequent, as a result of the bigger SSDs' sizes. Having bigger SSDs' sizes, it takes longer to fill them.



(a) BTIO (b) Madbench

Figure 4.6: Ups (active state - 1) and downs (standby state - 0.5) in the course of time, of a single disk drive, when the write buffering policy is applied. SSDs are 4 GB sized.

The Madbench case (Figure 4.4(b)) is slightly different. Reads and writes are interleaved, but also, after writing some data, the same previously written data are again read. In this specific case, around 300 MB of data are written, and that is the minimum size of a cache for this application, to start overwriting data without causing read misses. So, here, again, the usage of SSDs of at least 2GB becomes more than adequate. Figure 4.5(b)

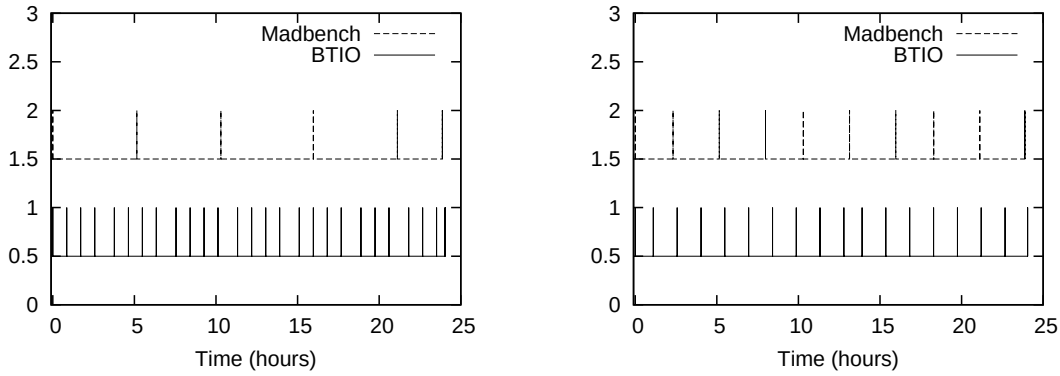
shows a time line of the effects in disk drives, for the MadBench application. The figure shows the ups (active state - 1) and downs (standby state - 0.5) in the course of time, of a single disk drive, when the proposed write buffering policy is applied. In this case, SSDs are 2 GB sized. All the showed ups are due to the fact that SSDs get filled, and contents must be written to disks. As the written blocks are less than in the BTIO application, ups are also less. Figure 4.6(b) shows the ups and downs in a time line when SSDs are 4 GB sized. As can be seen, ups are more less frequent, as a result of the bigger SSDs' sizes.

4.3.1 SSD partitioning

Some modern supercomputers such as Blue Gene and Cray, let applications execute on storage subsystems in a dedicated way. Unfortunately, that is not possible in other supercomputers where applications run on a concurrent fashion. In order to ensure that our power-saving aware write-buffering techniques are also satisfied in concurrent environments, we propose *SSD partitioning*, where for every SSD device, each application corresponds to a dedicated partition. So, like in the previous section, each application applies the write buffering policy to its own partition. When for any of the applications, the partition is filled, the disk is spinned up, and the contents, from that specific partition, written back to the disk. When doing this, it may happen that for one or several applications, their corresponding partitions are not filled. That is due to the fact that applications have different speeds when redirecting requests to SSDs, and the ones which write more data in less time usually fill their partitions. Figure 4.7(a) shows the ups (active state - 1/2) and downs (standby state - 0.5/1.5) in the course of time, of a single disk drive, when the proposed SSD partitioning policy is applied. The two previously described applications (BTIO and Madbench) are executed concurrently using a 4 GB sized SSD. Each application redirects its data to an equally 2 GB sized part. As the amount of written data are not the same in both applications, frequency of spin ups are not either the same. Thus, Madbench causes less ups than BTIO. The total amount of ups in the disk is the addition of ups from both applications.

Figure 4.7(b) shows the ups (active state - 1/2) and downs (standby state - 0.5/1.5) in the course of time, of a single disk drive, when the proposed SSD partitioning policy is applied. In this case, each application does not redirect its data to an equally sized part. BTIO redirects its data to a 3 GB sized part, while Madbench redirects its data to a 1 GB part. Here, the addition of the total amount of ups turns out to be less than in the equally sized configuration.

As we will show in our evaluation, setting up equal partition sizes still results in monetary costs improvements. As was previously said, while executing concurrently, for each spin-down period not always the same applications cause their partition capacity fills. As a result, we think that a dynamic partitioning policy, on execution time, can be established and may mean more monetary cost improvements. That dynamic policy specially would be easy to apply when applications just write data, and those data do not have to be read again. That is because, when moving partition boundaries, future read data in a certain partition, may be overwritten by new data corresponding to a new partition, causing disk spin ups for read requests of previously overwritten data.



(a) Equally sized parts

(b) Non equally sized parts

Figure 4.7: Ups (active state - 1/2) and downs (standby state - 0.5/1.5) in the course of time, of a single disk drive, when the SSD partitioning policy is applied. SSDs are 4 GB sized.

4.4 Power saving-aware prefetching policies

In this section, we describe several approaches included in the Prefetching module from the SSD-PASS architecture.

As was said in the previous section, some HPC applications exhibit repetitive behaviours, in which previously written data are read again. This characteristic makes optimal the previously described write buffering policy, because disk drives are only spun up when SSDs are filled, and not because some read data cannot be found on the SSDs. However, other applications present read accesses to data that cannot be found on the SSDs. That is because, those data have not been previously written, or they have been overwritten by another data. With the purpose of avoiding this situation, a specific area can be reserved in the SSDs, with an aim to host data that are supposed to be accessed in the future. This helps to remove unnecessary spins-ups. In order to make the most of that area, for power saving purposes, several prefetching policies are proposed on this thesis.

The objective of a conventional prefetching technique is to improve file access performance by issuing file data requests in advance. However, in this thesis we propose a power-saving aware prefetching which focuses on reading in advance, enough blocks in order to avoid break long-idle intervals. If a workload is sequential enough, reading successive blocks into the SSD device using a single I/O operation, can considerably satisfy almost all of read requests during a long idle period of time. Moreover, it can take advantage of shorter service demands that come from the not extreme head displacements at disk drives.

The prefetching policy is enforced at the SSD and consists of: deciding which blocks to prefetch (prediction), monitoring the block prefetching, and moving the prefetched blocks from the SSD-based cache to applications. In this thesis, we propose several adaptive prefetching policies as energy-efficient strategies, which are integrated in the SSD-based hybrid architecture presented in Section 4.2. For some policies, The prefetching module adjusts parameters in real-time such as window sizes, in order to obtain the best perfor-

mance for data-intensive applications.

4.4.1 Sequential read-ahead policy

This mechanism involves reading in advance, consecutive data blocks when a read miss occurs. We employ this approach by using adaptive window sizes. Depending on the workload access pattern, the policy chooses an appropiate window size to get the highest possible hit ratio. The prefetching policy dynamically calculates the maximum sequential data flow with the arrived requests, and uses it as the window size. We constantly derive spatially adjacent requests from the data flow, targeting each disk. Multiple simultaneous data flows or strided data flows make the derivation and modeling of sequentiality difficult. Merged strided data flows are identified by comparing each request block, not only against the nearly previous request block and length, but also to the n previous requests. We introduce a history-based log of the last n I/O accesses. This can be considered as augmenting the read-ahead window for this calculation from 1 to n. For every I/O access, if necessary, we linearly look through this record and find out if we have a higher sequentiality.

Algorithm 6 Calculation of the degree of sequential read-ahead

Block *reqBlock* is requested:

```
1: size ⇐ reqBlock - prevBlock
2: currBlock ⇐ reqBlock + (rwSize)/blockSize − 1
3: if size ≠ 1 then
4:    for i ⇐ separation.begin() to separation.end() do
5:       if reqBlock - i = 1 then
6:          prevBlock ⇐ i
7:          size ⇐ 1
8:       end if
9:    end for
10: end if
11: if size = 1 then
12:    separation.add(currBlock)
13:    if separation.size() > windowSize then
14:       separation.remove(min(separation))
15:    end if
16:    dataflows[currBlock] ⇐ dataflows[prevBlock]
17:    dataflows.erase(prevBlock)
18:    separ ⇐ currBlock - dataflows[currBlock]
19:    if maxSeq < separ then
20:       maxSeq ⇐ separ
21:    end if
22: end if
23: prevBlock ⇐ currBlock
```

The calculation of the degree of sequential read-ahead is described in Algorithm 6. Every time a *reqBlock* is requested, the distance (*size*) with respect to the previous block

Algorithm 7 Algorithm for sequential prefetching - Sequential read-ahead

Block *reqBlock* is requested:

```
1:  miss ⇐ 0
2:  for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1) do
3:    if i ∉ CACHE then
4:      miss ⇐ 1
5:    end if
6:  end for
7:  if miss = 1 AND size = 1 then
8:    for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1 + maxSeq) do
9:      putInCache(i)
10:   end for
11: else
12:   for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1) do
13:     putInCache(i)
14:   end for
15: end if
```

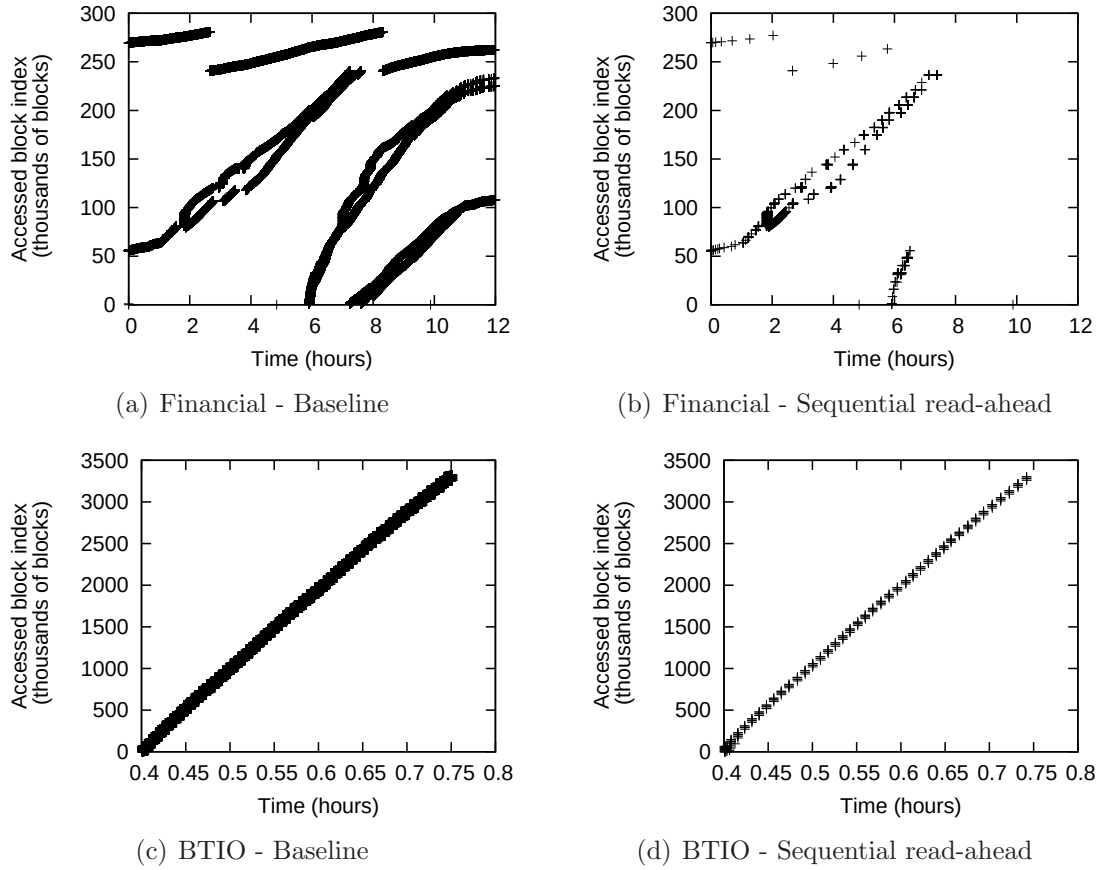
accessed (*prevBlock*) is determined (line 1). If blocks are not sequential, the system provides a second chance to find sequentiality. Lines 3-10 deal with the case of comparing *reqBlock* with the previous *windowSize* requests hosted in the *separation* history list. If *size* results 1 in this case, last accessed block in the current request (*currBlock*) is saved into the *separation* history list, and the least recently added block is removed, if the list is full (lines 11-15). Lines 16-22 record *currBlock* as the last block accessed in its data flow and the length of the data flow is compared with the length of the longest data flow (*maxSeq*) and updated if applicable. Line 23 records the last accessed block in the current request (*currBlock*) as the previous accessed block (*prevBlock*) for the next request.

Only when a read miss occurs and the distance with respect to the previous accessed request (*size*) turns out to be 1, the system prefetches a total of $seqMax$ blocks (shown in Algorithm 7).

The effectiveness of this approach can be seen in Figure 4.8. It shows the distribution of accessed disk requests in the course of two applications executions. The left part of the figure shows disk accesses without applying any prefetching policy. The right part of the figure shows disk accesses, by applying the previously described prefetching policy.

The policy is evaluated only for read accesses, and, in the specific case of the figures, it is not combined with the previously described write buffering policy. That means that read requests cannot be serviced by the writing policy parts of SSDs. By doing this, the real effect of the prefetching policy becomes more visible. The two applications present a high degree of sequentiality, so disk accesses are reduced.

As can be seen, the usage of the prefetching policy avoids disk accesses. However, the number of avoided disk requests are not so many, for the resulting idle times to be big enough. Also, the avoided disk requests are not concentrated enough in time to make idle times bigger. Instead, they are spread in the course of the applications execution. In spite



(a) Financial - Baseline

(b) Financial - Sequential read-ahead

(c) BTIO - Baseline

(d) BTIO - Sequential read-ahead

Figure 4.8: Distribution of accessed disk requests in the course of two applications executions. Sequential read-ahead policy has been applied (right).

of that, this approach has several advantages:

- Not all the misses cause data prefetching. Data are prefetched only when a miss happens because the current prefetched flow is not big enough, and must be extended. This avoids that subsequent requests wait very often for the prefetching stage to finish.
- The current flow can be extended as far as its size plus the maximum sequential flow size. If an application is very sequential, many read misses can be avoided, making idle times longer.

This approach has also several drawbacks:

- When the maximum sequential flow size is very big, subsequent requests must wait until the current prefetching stage to finish reading from disk.
- It does not avoid disk accesses in non-sequential applications.

4.4.2 Block sieving read-ahead policy

This approach is an enhancement of the algorithm presented in Section 4.4.1. In the previous approach, only when a read miss occurs and the distance with respect to the previous accessed request ($size$) is 1, the system prefetches a total of $seqMax$ blocks, which is the size of the maximum accessed sequential flow. However, when a read miss occurs, and the distance with respect to the previous accessed request is not 1, the system does not apply prefetching.

Here, when a miss occurs, and the distance with respect to the previous accessed request is not 1, not just the requested blocks are put into the cache. Also, blocks that are between the requested blocks and the closest flow stored in cache are prefetched, thus, extending the closest flow stored in cache. This makes that not only strictly sequential accesses cause prefetching. However, not all the closest flows to the current request can be extended. If the distance from the current request to the closest flow is bigger than a certain value, the flow better not be extended, because the blocks to prefetch are too many, and the subsequent requests could wait for too long.

Algorithm 8 Calculation of extension to the closest flow

Block $reqBlock$ is requested:

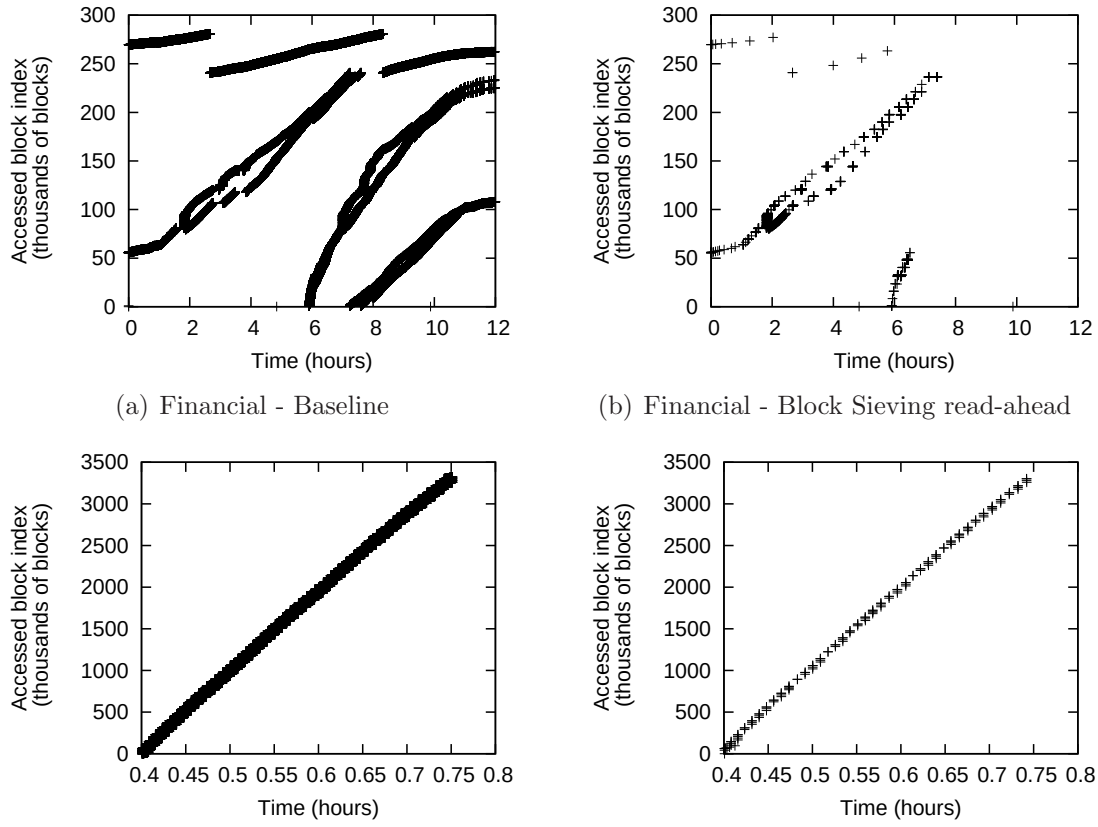
```
1:  min ⇐ MAXINT
2:  minAbs ⇐ MAXINT
3:  for i ⇐ flows.begin() to flows.end() do
4:    if reqBlock - i < min then
5:      min ⇐ reqBlock - i
6:      if min < 0 then
7:        minAbs ⇐ (reqBlock - i) * (-1)
8:      else
9:        minAbs ⇐ reqBlock - i
10:     end if
11:     if minAbs < 400 then
12:       if min > 0 then
13:         flows[reqBlock + rwSize/blockSize − 1] ⇐ i
14:         flows.erase(i)
15:         begin ⇐ reqBlock + rwSize/blockSize − 1
16:       else
17:         flows[i] ⇐ reqBlock
18:         begin ⇐ i
19:       end if
20:     end if
21:   end if
22: end for
```

The calculation of the extension of the closest flow is described in Algorithm 8. Every time that a list of blcoks ($reqBlock$) is requested, the distance ($minAbs$) to all the stored flows is determined (lines 1-10). If that distance ($minAbs$) is less than a certain value (line

11), the flow is extended, on the right (lines 12-15), when the current request is closer to the end of the flow (i), or on the left (lines 16-19), when the current request is closer to the beginning of the flow ($flow[i]$).

Only when a read miss occurs and the distance with respect to the previous accessed request ($size$) is not 1, the system prefetches the total number of missing blocks in the previously extended flow (shown in Algorithm 9; lines 12 - 14).

The effectiveness of this approach can be seen in Figure 4.9. It shows the distribution of accessed disk requests in the course of the executions of two applications. The two applications present a high degree of sequentiality, so disk accesses are reduced. The left part of the pictures shows disk accesses without applying any prefetching policy. The right part shows disk accesses, when the described prefetching policy is applied.



(a) Financial - Baseline

(b) Financial - Block Sieving read-ahead

(c) BTIO - Baseline

(d) BTIO - Block Sieving read-ahead

Figure 4.9: Distribution of accessed disk requests in the course of two applications executions. Block Sieving read-ahead policy has been applied (right).

As can be seen, the application of this policy does not improve in a high degree the previous policy. It just avoids some more request accesses to the disk drive, but it does not make much bigger idle times.

Algorithm 9 Algorithm for sequential prefetching - Block Sieving read-ahead

Block *reqBlock* is requested:

```
1:  miss ⇐ 0
2:  for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1) do
3:    if i ∉ CACHE then
4:      miss ⇐ 1
5:    end if
6:  end for
7:  if miss = 1 AND size = 1 then
8:    for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1 + maxSeq) do
9:      putInCache(i)
10:   end for
11: else
12:   for i ⇐ flows[begin] to (begin) do
13:     putInCache(i)
14:   end for
15: end if
```

4.4.3 Extended Window read-ahead policy

This approach is an extension of the previous algorithm. Here, a lot more data blocks are prefetched. As can be seen in Algorithm 10, every time a *reqBlock* is requested, the size of the flow in cache it belongs to is calculated. This parameter is the next number of blocks to prefetch (see Algorithm 11).

Algorithm 10 Calculation of distance to the next flow

Block *reqBlock* is requested:

```
1: for i ⇐ flows.begin() to flows.end() do
2:   if prevBlock <= i AND prevBlock >= flows[i] then
3:     sepa ⇐ i − flows[i] + (rwSize)/blockSize
4:   end if
5: end for
```

The effectiveness of this approach can be seen in Figure 4.10. It shows the distribution of accessed disk requests in the course of the executions of two applications. The two applications present a high degree of sequentiality, so disk accesses are reduced. The left part of the pictures shows the disk accesses without applying any prefetching policy. The right part shows disk accesses, by applying the described prefetching policy.

The application of this policy improves in a high degree the previous policy. The drawback of this approach is that a lot of blocks are prefetched, reducing the performance of the two applications. Also, a lot of space is wasted as a result of the great number of blocks that are prefetched.

Algorithm 11 Algorithm for sequential prefetching - Extended Window read-ahead

Block *reqBlock* is requested:

```
1: miss ⇐ 0
2: for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1) do
3:    if i ∉ CACHE then
4:       miss ⇐ 1
5:    end if
6: end for
7: if miss = 1 AND size = 1 then
8:    for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1 + sepa) do
9:       putInCache(i)
10:   end for
11: else
12:   for i ⇐ flows[begin] to (begin) do
13:      putInCache(i)
14:   end for
15: end if
```

4.4.4 Offline read-ahead policy

All the previous approaches did not get the most of the write buffering policies. In this approach an offline algorithm is proposed. This approach gets the most of it, and also, by taking into account the size of the cache, it adds and removes blocks from it, in order to provide the largest idle times.

The offline algorithm consists of: Reserving a part in the SSDs to host data blocks that are going to be read. The rest of the space is reserved for blocks that will be redirected to SSD as a result of the write buffering policy.

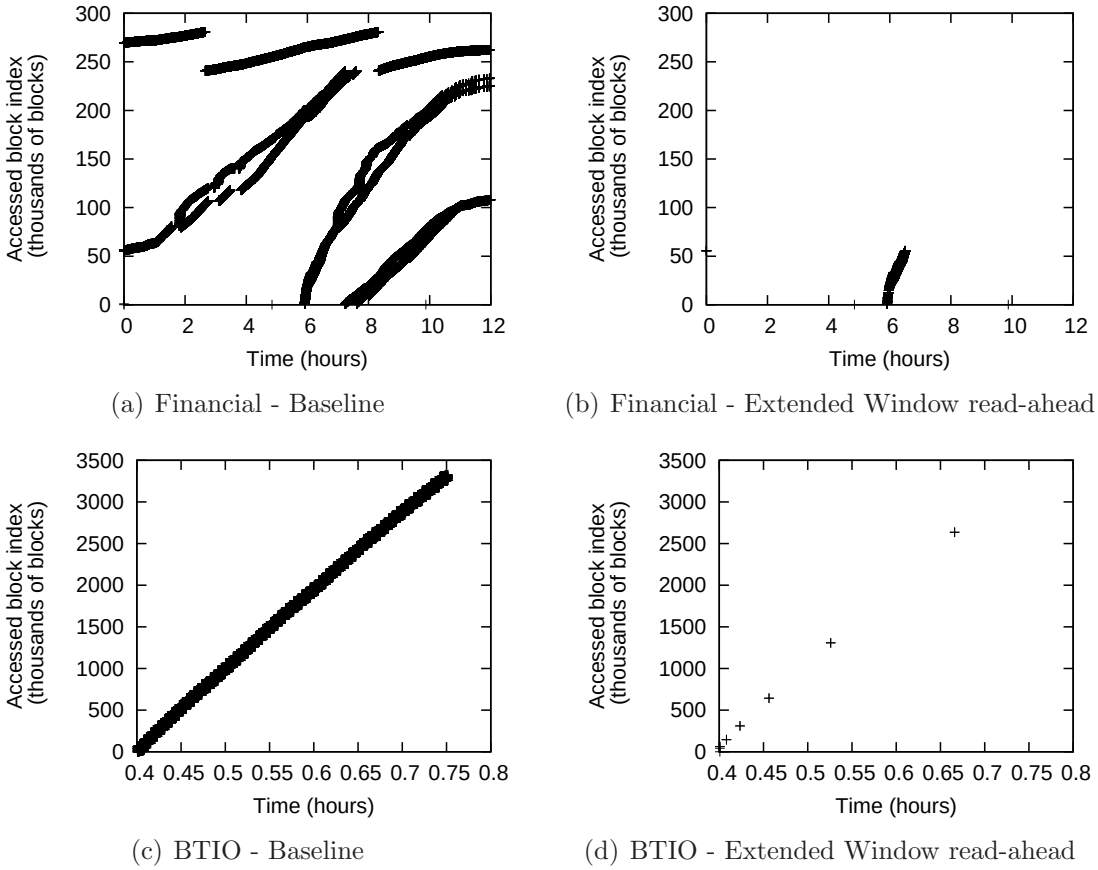
When a data block must be written, it goes to the write buffering part and keeps placed on it. Whenever a data block must be read, it checks the write part first, to see if the block has been written there, and if so, the block is read from there. If that block is considered to be accessed later, it is copied to the read part, for future accesses, as shown in Algorithm 12.

Algorithm 12

Block *reqBlock* is requested:

```
1: for i ⇐ reqBlock to (reqBlock + rwSize/blockSize − 1) do
2:    if i ∈ CACHE_w AND willBeAccessed(i) then
3:       putInCache_r(i)
4:    end if
5: end for
```

This algorithm is useful both for sequential and not sequential workloads. Some applications does not present a clear sequentiality, so it becomes difficult to predict future read accesses, by learning dynamically, from previous sequential patterns.



(a) Financial - Baseline

(b) Financial - Extended Window read-ahead

(c) BTIO - Baseline

(d) BTIO - Extended Window read-ahead

Figure 4.10: Distribution of accessed disk requests in the course of two applications executions. Extended Window read-ahead policy has been applied (right).

The algorithm learns, from previous executions, which blocks to prefetch on the SSDs. In the learning executions, blocks are read from the write buffering part as much as possible. After the learning phase, missing blocks are analyzed, in order to know which blocks should be prefetched before the application starts.

In order to detect block areas that has been accessed in the course of the applications executions, a new algorithm is proposed (see Algorithm 13). This algorithm helps to decide which areas from disk drives should be prefetched before application execution, in order to provide the biggest idle times.

This algorithm only detects bands when the read accesses are spread in the course of the application execution and all of them are concentrated in specific areas of the disk drive.

As can be seen, every read block that has been missed from cache is checked to see if it belongs to any band of the disk (line 3). If the difference with respect to the minimum block of the band (*min*) is bigger than the *bandSize*, or there exists a gap between the actual checked block (i) and the previous one (*prevLBA*), a new band is selected, when the number of *LBAs* in that band is higher than a certain threshold, *LBAsThreshold* (lines 4

Algorithm 13 Band detection method

```
bandDetection()
 1: min ⇐ CACHE.begin() , max ⇐ CACHE.end()
 2: prevLBA ⇐ min
 3: for i ⇐ CACHE.begin() to CACHE.end() do
 4:    difference ⇐ i − min
 5:    if (difference < bandSize)&(i − prevLBA < maxGap) then
 6:       max ⇐ i , LBAs + +
 7:    else
 8:       if LBAs ≥ LBAsThreshold then
 9:          band[j].min ⇐ min, band[j].max ⇐ max
10:          j + +
11:       end if
12:       min ⇐ i , max ⇐ i
13:       LBAs ⇐ 0
14:    end if
15:    prevLBA ⇐ i
16: end for
```

- 13).

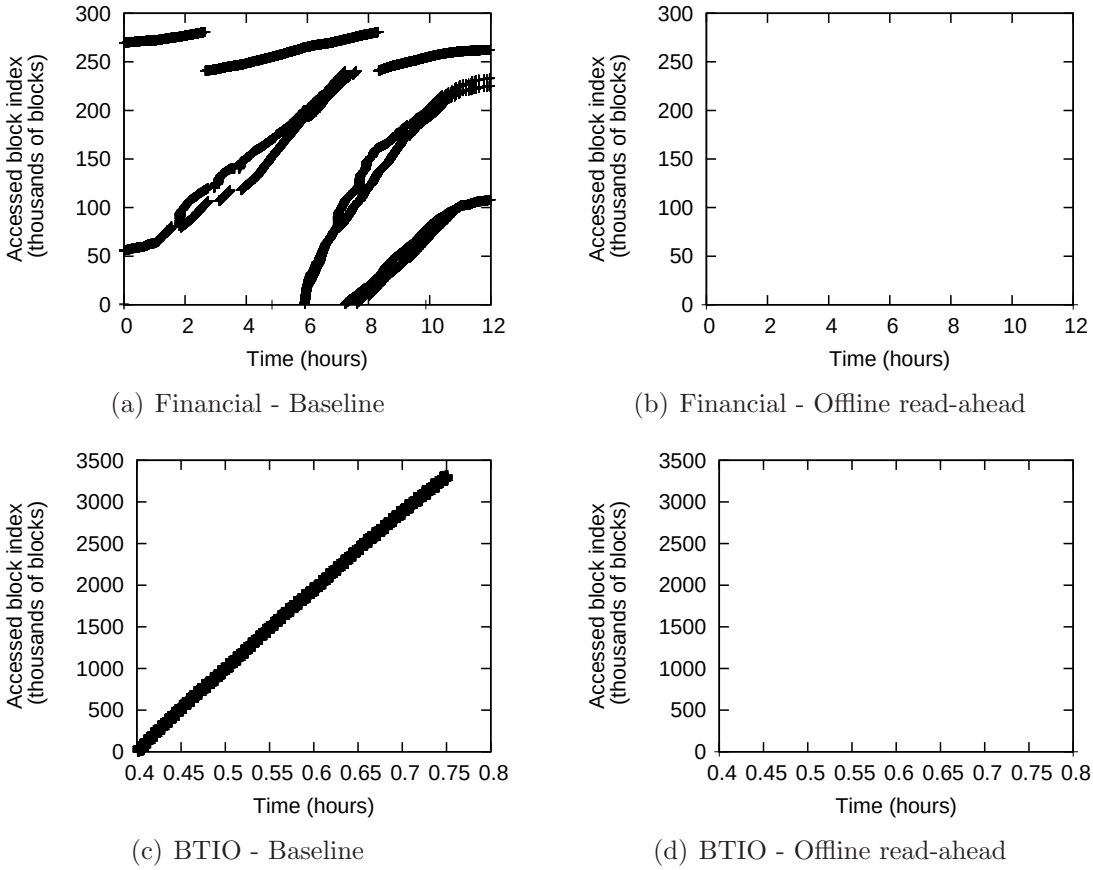
Once specific areas have been detected, before subsequent executions, data in bands are moved to the read part of SSDs (Algorithm 14).

Algorithm 14

```
1: for i ⇐ 0 to (band.size() − 1) do
2:    for j ⇐ band[i].min to (band[i].max) do
3:       putInCache_r(j)
4:    end for
5: end for
```

The usage of this algorithm has several advantages with respect to previously described ones:

- It can be used both for sequential or not sequential applications.
- As the prefetched blocks are moved from disk to SSDs before starting the execution of the applications, currently served requests do not have to wait for previous requests to be served.
- As read misses are known before the execution of the applications start, disk drives do not have to spend time in being waken up, and then, the missed requests do not have to wait for being serviced.
- As we will show in our evaluation, space in SSDs is not wasted, because just the needed blocks are moved from disk to SSDs. Blocks that are not going to be used are not moved.



(a) Financial - Baseline

(b) Financial - Offline read-ahead

(c) BTIO - Baseline

(d) BTIO - Offline read-ahead

Figure 4.11: Distribution of accessed disk requests in the course of two applications executions. Offline read-ahead policy has been applied (right).

The effectiveness of this approach can be seen in Figure 4.11.

4.5 Economic model

In this section, we describe an economic model in order to evaluate if our proposed storage power saving-aware approaches are justifiable and feasible. The model compares an architecture deployed with magnetic disks, to our hybrid storage architecture consisted of disks and SSDs devices. Initially, costs of acquisition of the first architecture may be higher than costs of acquisition of the second one. As time goes on, the investment in our power-saving architecture may be returned because cost of power in the disk subsystem is a significant element to consider. In order to show how this investment is returned, we turn both the cost of acquisition and the power into monetary terms. Both disk and SSD devices have a limited life-span. The model also takes into account device replacements costs as a consequence of their use. We identify two principal SSD solutions, MLC and SLC. In this thesis we present results for SLC-based SSD due to this model present better benefits than MLC in terms of performance and durability.

The acquisition cost of a SATA HDD per GB is around 0.14 USD, while the cost for a SLC-based SSD per GB is around 9.2 USD [SPBW10]. Electricity prices vary all over the world. An average of electricity tariffs of industrialised countries of Europe and USA is 12 US cents/kWh[ele11a, ele11b], although some factors are causing electricity prices increases [BCFP+06] in many countries.

Disk devices specifications provide two important metrics about their durability: mean time to failures (MTTF) and constant start stop cycles (CSS). MTTF estimates the number of hours that a disk will last until the next failure. This metric is based on the assumption that disks are powered on 100% of the time. Moreover, several studies show that there is a correlation between MTTF and some parameters generally believed to impact durability/reliability [Sik07, PWB07, GPW10], what make it diffuse to use as a disk durability metric. CSS are the maximum number of start/stop cycles that disk devices are designed to handle during their service life. The life of disk devices is usually designed for about 30,000 - 50,000 cycles. If this value is reached for a disk device, it means that a lot of wear has occurred to the head and also to other components such us the spindle motor, making it unusable.

Every time magnetic disk or SSD devices need to be replaced because they have exceeded their durability metrics, new acquisition costs are added to the total monetary costs. It is important to note that in SSD devices, blocks are erased before re-writing them again, and the maximum number of erasures before devices become unusable is limited. MLC provides higher densities per cell and the memory cells have to be re-written in larger chunks. SLC provides lower densities and last longer. In MLC devices the durability is limited to 10,000 erasures per block while in SLC devices durability is limited to 100,000 erasures per block [SPBW10].

In order to provide an estimation for the energy costs of the proposed approaches, the energy consumption of both disk and SSD devices has been modeled. The power model for the disk devices employs an extension of the 2-Parameter Model described in [ZSG+03]. This model applies the following formula to calculate the energy consumption estimation of each disk:

$$E_{disk} = E_{activeDisk} + E_{idleDisk} + E_{standbyDisk} \tag{4.1}$$

Where $E_{activeDisk}$ is calculated as $P_{activeDisk} x T_{activeDisk}$, $P_{activeDisk}$ is the power consumption when the disk is active, and $T_{activeDisk}$ is the time spent by the disk while satisfying disk requests. $E_{idleDisk}$ is calculated as $P_{idleDisk} x T_{idleDisk}$, $P_{idleDisk}$ is the power consumption in the idle mode, and $T_{idleDisk}$ is the length of the idle period. $E_{standbyDisk}$ is calculated as $P_{standbyDisk} x T_{standbyDisk}$, $P_{standbyDisk}$ is the power consumption in the standby mode and $T_{standbyDisk}$ is the length of the standby period.

As SSDs have different energy consumption ratios, we provide a specific SSD power model, in which consumption estimation is given in the following formula:

$$E_{SSD} = E_{activeSSD} + E_{idleSDD} \tag{4.2}$$

Where E_{active} is calculated as $P_{activeSSD} x T_{actoveSSD}$, $P_{activeSSD}$ is the power consumption when the SSD is active, and $T_{activeSSD}$ is the time spent by the SSD while satisfying

SSD requests. $E_{idleSSD}$ is calculated as $P_{idleSDD}xT_{idleSSD}$, $P_{idleSSD}$ is the power consumption in the idle mode, and $T_{idleSDD}$ is the sum of the idle periods.

In each evaluation, the model obtains energy consumptions for periods of 24 hours. An estimation of the monetary costs for those consumptions, in the $i-th$ storage element, is given by the following formula:

$$Costs_{1Day\_i} = (E_{disk\_i} + E_{SSD\_i}) * 3.33x10^{-8} \tag{4.3}$$

Where $3.33x10^{-8}$ is the price in \$/Ws. Units of energy are given in Ws. An estimation of the monetary costs for periods of 1 month is given by the formula:

$$Costs_{1Month\_i} = Costs_{1Day\_i} * 30 \tag{4.4}$$

The model analyzes the cost and the amortization in a seven year period. We chose that frame of time, considering that, typically, wear in disk drives is high after years 5-7 of operation under normal conditions [Sik07]. The formula for that calculation:

$$Costs_{7Year\_i} = Costs_{1Month\_i} * 84 \tag{4.5}$$

In order to know the months that a single disk can last, before being replaced, the number of start/stop cycles in a period of 24 hours must be known. Thus, the calculation of the duration of a single disk drive is given by the following formula:

$$Duration_{disk\_i} = (CSS/startStop\_i)/30 \tag{4.6}$$

Where $startStop\_i$ is the number of start/stop cycles in a period of 24 hours, and CSS is the maximum number of start/stop cycles that the disk can handle during its service life.

In order to know the months that a single SSD can last, before being replaced, the maximum number of written GB per day must be known. The calculation of the duration of a single SSD is given by the following formula:

$$Duration_{SSD\_i} = (100,000/(GB_{perDay\_i}/GB_{perSSD}))/30 \tag{4.7}$$

Where GB_{perDay_i} are the written GB in a day, and GB_{perSSD} is the size in GB of the SSD under review. 100,000 is the maximum number of erasures per block in the SSD.

The price per disk, for a disk of a certain size, is given by the formula:

$$Price_{disk} = GB_{perDisk} * 0.14 \tag{4.8}$$

The price per SSD, for a disk of a certain size, is given by the formula:

$$Price_{SSD} = GB_{perSSD} * 9.2 \tag{4.9}$$

Finally, total monetary costs for our storage power aware approach are represented by the formulas:

$$Costs_{Repl\_i} = Price_{disk}*(1+84/Duration_{disk\_i})+Price_{SSD}*(1+84/Duration_{SSD\_i}) \quad (4.10)$$

$$Costs_{SSD\_PASS} = \sum_{i=1}^{NSE} (Costs_{Repl\_i} + Costs_{7Year\_i}) \quad (4.11)$$

Where $Costs_{Repl\_i}$ represents acquisition and replacement costs in a seven year period, for the $i-th$ storage element. NSE is the total number of storage elements that our power aware architecture has.

Total monetary costs for a baseline architecture, deployed with magnetic disks, are given by the formula:

$$Costs_{Baseline} = \sum_{i=1}^{NSE} (Price_{disk} + E_{disk\_Baseline\_i} * 3.33x10^{-8} * 30 * 84) \quad (4.12)$$

Where $E_{disk\_Baseline\_i}$ is the energy consumption of the $i-th$ magnetic disk in a 1 day period.

Finally, monetary costs savings are given by the formula:

$$CostsSavings = Costs_{Baseline} - Costs_{SSD\_PASS} \quad (4.13)$$

4.6 Summary

In this chapter, two new techniques have been proposed in order to save power in HPC applications. On the one hand, we have proposed a new write buffering policy, which gets the most of SSD drives. It also gets the most of repetitive patterns in HPC applications, in order to avoid unnecessary spin-ups. On this same context, an SSD-patitioning policy is proposed. It gets the most of several concurrent applications in order to avoid disk spin-ups, and save power.

On the other hand, as not every HPC application present repetitive patterns, not all the read requests can be found in SSDs, causing disk spin-ups. Four prefetching algorithms are proposed in order to solve that problem.

Finally, an economic model was proposed, in order to evaluate the previously mentioned approaches, in the long term.

The proposed approaches and their experimental results have been published in:

- *Power Saving aware prefetching for SSD-based systems.* Laura Prada, Javier Garcia, J. Daniel Garcia, Jesus Carretero. The Journal of Supercomputing. Springer. March, 2011. Impact Factor: 0.687.

- *A Power-aware Based Storage Architecture for High Performance Computing.* Laura Prada, Javier Garcia, J. Daniel Garcia, Jesus Carretero, Alberto Nuñez. 13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011). September, 2011.

- *Using Write Buffering and Read Prefetching Between Flash and Disk Drives to Save Energy in an Hybrid System.* Laura Prada, J. Daniel Garcia, Jesus Carretero. 16th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010). July, 2010.

- *Power Saving-aware Solution for SSD-based Systems.* Laura Prada, J. Daniel Garcia, Jesus Carretero, Javier Garcia Blas. International Conference on Mathematical Methods in Science and Engineering (CMMSE 2010). Almeria, Spain. June, 2010.

- *Saving power in flash and disk hybrid storage system.* Laura Prada, Jose Daniel Garcia, Jesus Carretero, and Felix Garcia. 17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'09). London, England. September, 2009.

- *Ahorro energetico en un sistema de almacenamiento hibrido compuesto por un disco duro y varias memorias flash.* Laura Prada, J. Daniel Garcia, Jesus Carretero, and Felix Garcia. Actas de las XX Jornadas de Paralelismo. La Coruña, Spain. September, 2009.

Chapter 5

Evaluation

The previous chapters proposed a method to construct black box models, for disks drives, based on probabilistic distributions. Also, a generic power aware I/O architecture and techniques to save power are proposed. This chapter focuses on the evaluation of the proposed methods and approaches.

The chapter is organized in three sections. First, in Section 5.1 we evaluate several characteristics for the construction of black box models, presented in Chapter 3. Second, evaluation of the different policies presented in Chapter 4 are described in Section 5.2. Third, in Section 5.3 the black box model approaches are analyzed for their application on the modeling of power aware I/O architectures.

5.1 Black box modeling Evaluation

In this section, we evaluate the proposed method in Chapter 3. The section is organized in four subsections. In Subsection 5.1.1 we show validation results of our service time measurement tool, *play*. For its validation, we have used two synthetic traces. Evaluation of models construction, as well as comparisons with DiskSim, in terms of accuracy and performance, is described is Subsection 5.1.2. Evaluation of already constructed models, by using non-trained traces, is presented in Subsection 5.1.3. The already constructed models are based on several real traces. Subsection 5.1.4 shows results for construction of models based on synthetic traces. We evaluate those based-on-synthetic-traces models by using real traces.

We have tested our method using a SCSI disk as beta test. The disk is a Seagate Cheetah 10K.7. Its main features are shown in table 5.1. We have constructed several black box models for the disk and validated the models using several scenarios, as shown in the previously cited subsections.

Specification	Value
Heads	2
Discs	1
Bytes per sector	512
Bytes per track	556 Kbytes
Default read/write heads	2
Spindle speed	10000 rpm
I/O data transfer rate	320 Mbytes/sec
Formatted capacity	73.4 Gbytes
Guaranteed sectors	143,374,744
Cache buffer	8 Mbytes
Average latency	3 msec

Table 5.1: Manufacturer specifications for disk ST373207LC

5.1.1 Measuring Service time

In this section we validate our service time measurement implementation, *play*, against *dxreplay*, a measurement service time tool, included in *DIXtrac* [SG00, SGLG02, BS02]. We compare both programs by feeding them with two synthetic traces, under different configurations.

Figure 5.1 shows the CDF (Cummulative Distribution Function) results of running the two synthetic traces on the Seagate Cheetah 10K.7 disk by using *dxreplay*, from *DIXtrac*, compared to runs on the same disk by using *play*, our service time measurement program.

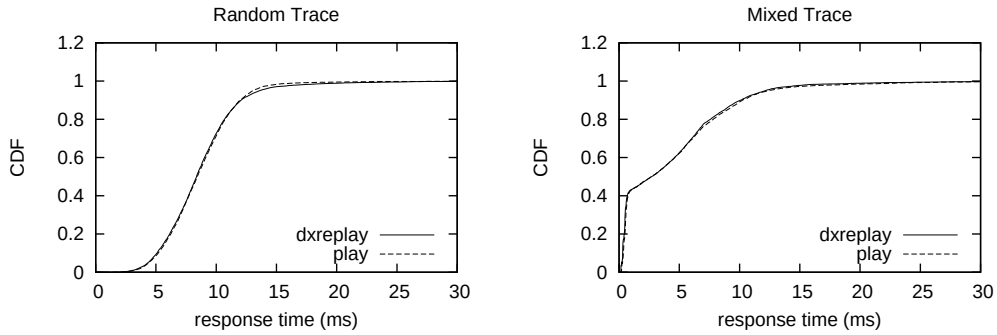


Figure 5.1: CDF's for response times from a Seagate Cheetah 10K.7 disk and two synthetic traces.

The trace, Random, has 10,000 requests, of which $\frac{2}{3}$ are reads and $\frac{1}{3}$ are writes. The LBNs are random and are distributed across the entire disk. The request size ranges from 1 KB to 8 KB. The other trace, Mixed, has 5,000 requests. $\frac{2}{3}$ of the total requests are reads and the rest are writes. 20% of the requests are sequential and 30% are local. The remaining 50% have random LBNs. The request sizes range between 1 KB and 8 KB.

We used the demerit [RW94] error as our metric to validate our service time measurement program, *play*. It is defined as the root mean square of the horizontal distances

between the distribution that comes from *dxreplay* and the distribution from our service time measurement program, *play*. We present it in absolute terms (as a difference in milliseconds), to compare it with other *DIXtrac* models [DIX12].

Random was run after deactivating the buffer cache to the disk drive. Mixed was executed after enabling again the cache, and hence, the read-ahead and immediate write reporting.

The demerit figures obtained for the Random and Mixed traces are, respectively, 0.59 ms and 0.47 ms. This is a quite good match, so this lead us to consider that *play* is validated and we use it in the rest of the evaluations of the chapter.

dxreplay incorporates a mechanism to restrict the maximum number of pending I/O requests that can be enqueued in the disk drive, due to the effect of bursty workloads. We also incorporated such a mechanism in our service time measurement program, *play*, by using mutexes. We compared again both measurement tools by executing the previously described synthetic traces, but making them more bursty. To notice the effect of the queuing limiting mechanism, we executed both traces by letting 1, 2, and 3 maximum enqueued requests.

Figure 5.2 shows the CDF results of running the new bursty Mixed trace on the Seagate Cheetah 10K.7 disk by using *dxreplay*, compared to the runs on the same disk by using *play*. The cache of the disk drive was activated. The demerit figures obtained for 1, 2, and 3 maximum enqueued requests are 1.3 ms, 3.1 ms, and 1.4 ms, respectively.

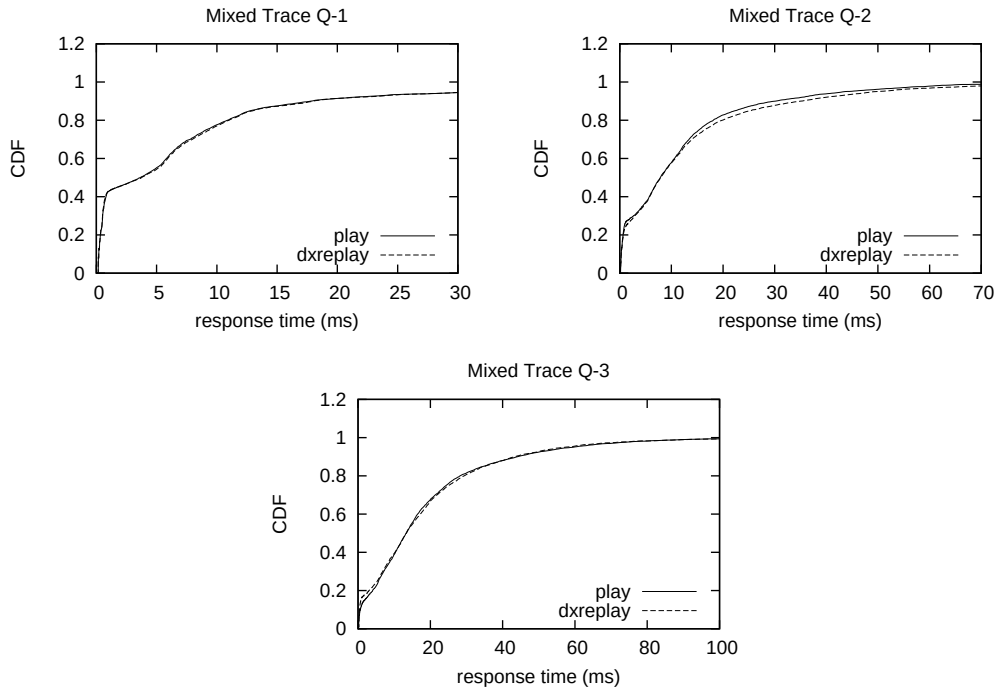


Figure 5.2: CDF's for response times from a Seagate Cheetah 10K.7 disk and a bursty Mixed trace. The maximum number of pending I/O requests that can be enqueued in the disk drive are limited to 1 (Q-1), 2 (Q-2), and 3 (Q-3)

Figure 5.3 shows the CDF results of running the new bursty Random trace on the Seagate Cheetah 10K.7 disk by using *dxreplay*, compared to the runs on the same disk by using *play*. The cache was deactivated in the disk drive.

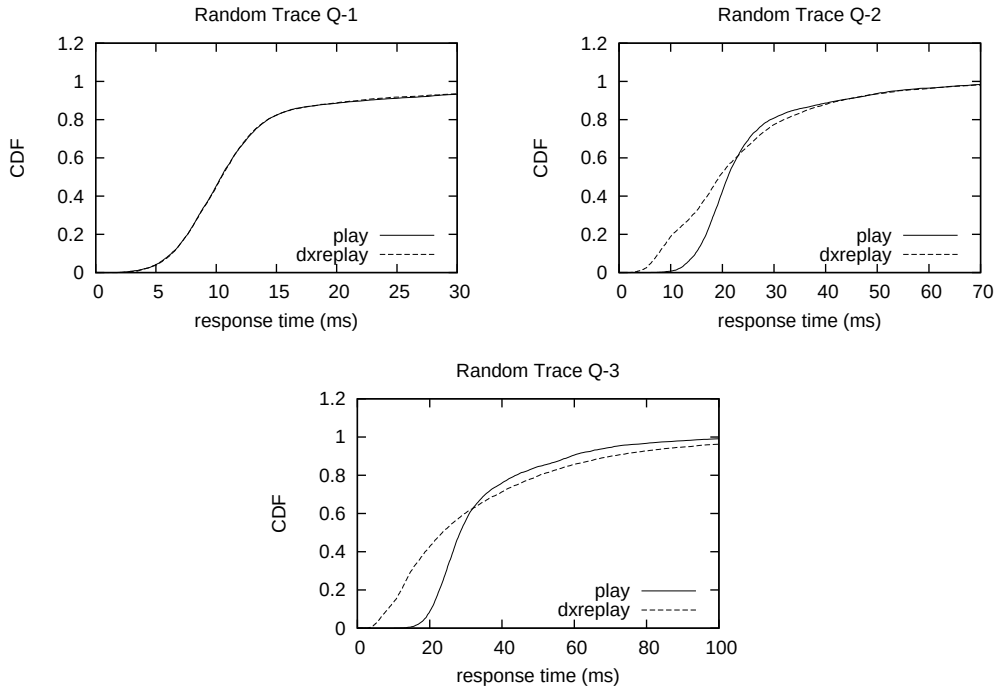


Figure 5.3: CDF's for response times from a Seagate Cheetah 10K.7 disk and a bursty Random trace. The maximum number of pending I/O requests that can be enqueued in the disk drive are limited to 1 (Q-1), 2 (Q-2), and 3 (Q-3)

The demerit figures obtained for 1, 2, and 3 maximum enqueued requests are 0.53 ms, 8.57 ms, and 12.29 ms, respectively. Note that with 2 and 3 maximum enqueued requests demerits increase significantly. That happens because in *play*, for our Seagate Cheetah 10K.7 disk, some specific LBNs or offsets from the Random trace are invalid in certain moments. By invalid, we mean that for our specific disk, the aggressive head movements or seekings, from some specific LBNs to anthers, produce errors. This fact has collateral effects: As some requests are not correctly measured, the subsequent requests may stay in the queue for longer, when the incorrectly measured requests last longer than they should. In the tests, for the Random trace, this brings on a shift on the right at *play* CDFs for the 2 and 3 maximum enqueued requests approaches.

As we can see, for the Mixed trace, the difference between the results obtained by using *play* and *dxreplay*, is not so noticeable. That is due to two main reasons: As the read-ahead and immediate write reporting were activated, some of the requests were serviced directly from the cache, avoiding being serviced from the platters, and hence, the head movements or seeks. Also, the Mixed trace is not as random as the Random trace, and some of the aggressive head movements or seekings are avoided.

However, for both traces, the obtained demerits are pretty good when the maximum

number of enqueued requests is 1. In both cases, when some requests are not correctly measured, the subsequent requests are not affected, because they start once the previous requests have finished. In the view of the obtained results, when the maximum number of enqueued requests is 1, we can say that the incorrectly measured requests are not many.

5.1.2 Constructing models

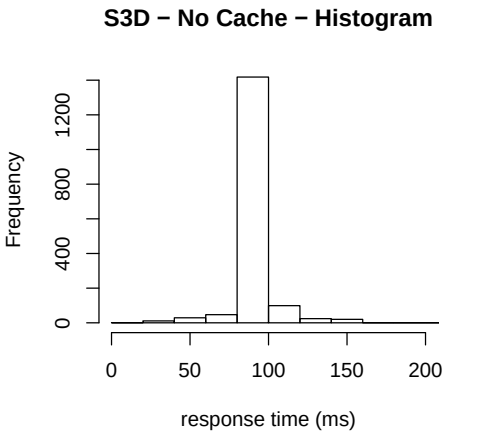
In this section we show the results obtained by applying our method, for predicting already trained traces. We first construct models by using some traces, and then, we predict response times from that models, by using the same traces we used to construct the models. So, for each of the 5 previously described workloads, we identify the obtained distributions, fit them to some known distributions, and figure the causes out for them. To see the effect of the disk buffers in the predictions, for each workload, we deactivated and activated the disk cache again. We analyzed each of the previously described traces to show the variety of behaviours a specific disk presents in the view of different traces.

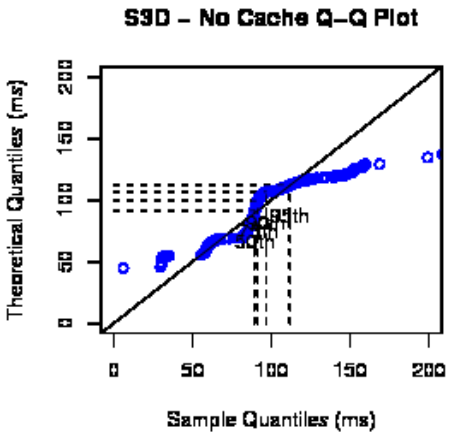
In each test, we plot the histogram for the data obtained by using *play* (our service time measurement program). By representing the histogram, we identified the possible distributions to fit to. After that, we fit the previously identified distributions to theoretical distributions, and show them in tables. Then, we find the causes for them, and on the basis of those causes, we construct the models. We replay the traces again on the recently constructed models and show the Q-Q plots to judge the goodness-of-fit of the models to the experimental data, and their CDFs to see how well they superimpose. The reason we analyze in such a detail every trace is to show the wide range of characteristics each specific trace presents on a specific disk.

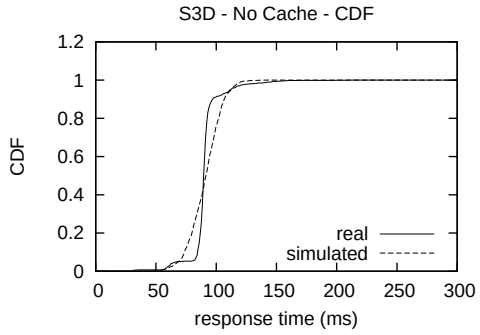
After analyzing each specific workload and its implications with and without the usage of the disk buffers, we compare our models against the DiskSim simulator [GWW+99]. We obtained the DiskSim model parameters for the real disk by running *DIXtrac* [SG00, SGLG02, BS02] on the Seagate Cheetah 10K.7 disk. As previously said, *DIXtrac* is a characterization tool for SCSI disk drives. It obtains more than 100 parameters to get a very detailed model of the disk, an those parameters are used by DiskSim.

We replay each of the 5 previously described workloads on DiskSim, and show their CDFs altogether with our models' and the real disk's. We also compare the demerit figures for each trace and both models, against the real disk. As in the previous specific analysis, we made the comparisons by deactivating first the disk buffers and then by activating them again.

We also make a comparison of the time it takes for both models to execute. We deactivate all the I/O and statistic processes in both simulators, and execute for each of the previously described traces. We also make a comparison in terms of speedup, to see how much faster than DiskSim our model is. Here, again, for each trace we make two comparisons: By deactivating the disk buffers, and by activating it again.





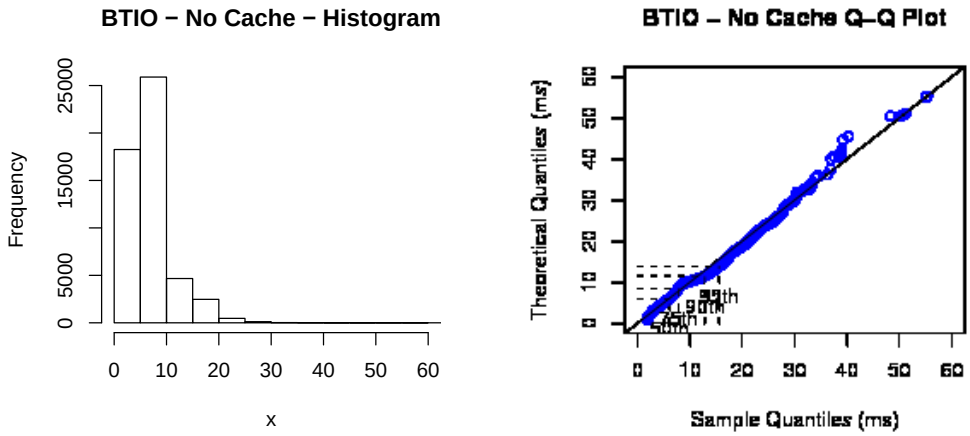


	type	μ	σ
Distribution 1	Normal	90.44	13.41

Figure 5.4: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a S3D workload. The table shows the parameters of the modeled distribution. Caching is not activated the disk drive

Figure 5.4 shows the histogram, Q-Q and CDF's plots, for the S3D workload. The usage of the disk cache was deactivated. We identified only one distribution, which parameters are shown in the figure. In this workload, although accesses are sequential, the size for each request is 2048 blocks. This makes response times are long. Also, as accesses are bursty, and most of the requests are enqueued, response times are even longer. So in this trace response times are dominated by enqueuing times, and their values are centered at the mean of the modeled distribution.

The trace also presents few short response times, as a result of the few idle periods of the trace. Idle periods avoid bursty accesses, in which most of the time a request must wait to be serviced until the end of the previous ones. Another reason for the short and also longest response times is the effect of the scheduling algorithm of the disk, which reorder the requests and the ones which came later (shortest) may be serviced before than the ones which came before (longest). In this trace, as we identified only one distribution we did not have to distinguish among several distributions and therefore, neither the reasons for them. All the response times will be generated from the same distribution. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 8%.

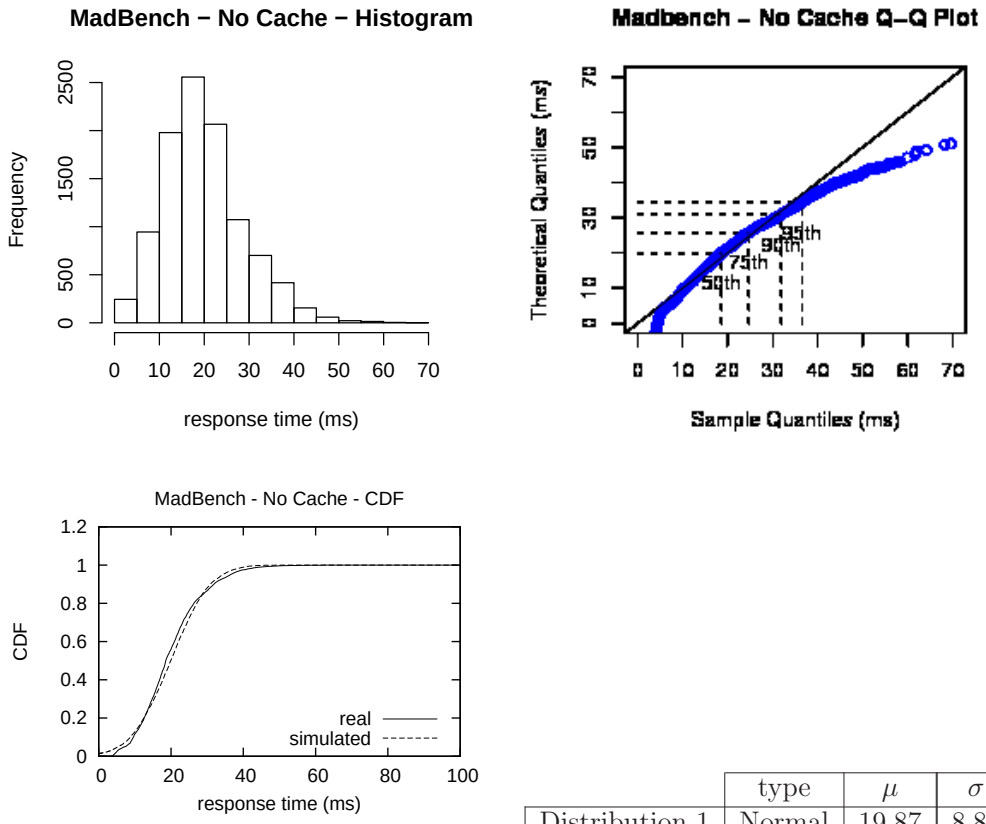


	type	μ	σ
Distribution 1	LogNormal	1.80	0.51

Figure 5.5: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a BTIO workload. The table shows the parameters of the modeled distribution. Caching is not activated in the disk drive

Figure 5.5 shows the histogram, Q-Q and CDF's plots, for the BTIO workload. The disk cache was deactivated. In this trace, we also identified one distribution, whose parameters are shown in the enclosed table. As a result of this, we did not either have to distinguish among different distributions and neither finding their reasons out. As in the previous trace, all the response times are generated from the same distribution.

The mean size of the request is 128 blocks and the accesses are also very sequential. However, the trace is not very bursty, making response times just dependant on the request size. Longer response times are due to the few bursty requests of the trace or the requests coming after some periods of idleness. As previously said, when a trace is very bursty, some requests must wait until the end of the previous ones, making their response times longer. For some specific disks, after a period of idleness the next request to be serviced takes longer than as usual. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 8.3%. Here, the disparity in the CDFs is due to the error when fitting the distribution.

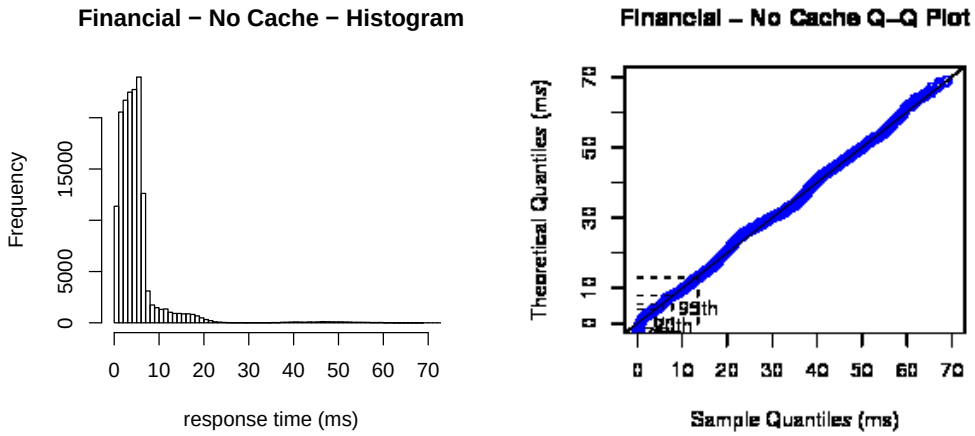


	type	μ	σ
Distribution 1	Normal	19.87	8.82

Figure 5.6: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a MadBench workload. The table shows the parameters of the modeled distribution. Caching is not activated in the disk drive

Figure 5.6 shows the histogram, Q-Q and CDF's plots, for the MadBench workload. We identified one distribution for this trace. As a result of this, we did not either have to distinguish among different distributions and neither finding their reasons out. As in the previous trace, all the response times are generated from the same distribution. Since it is extremely bursty, response times are dominated by enqueuing times. Also, requests are not very sequential, making longer response times.

The mean size of requests is 256 blocks. The scheduling algorithm produces response times shorter than the mean of the distribution, and also do the few periods of inactivity. In a bursty workload, the scheduling algorithm reorders the requests and the ones which came later (shortest) may be serviced before than the ones which came before (longest). Also, in a bursty workload, the periods of inactivity let some requests to be serviced without having to wait for the previous ones. The cache was deactivated. In the shown Q-Q and CDF's plots can be seen the goodness-of-fit of the model. The relative mean error is 8.4%.

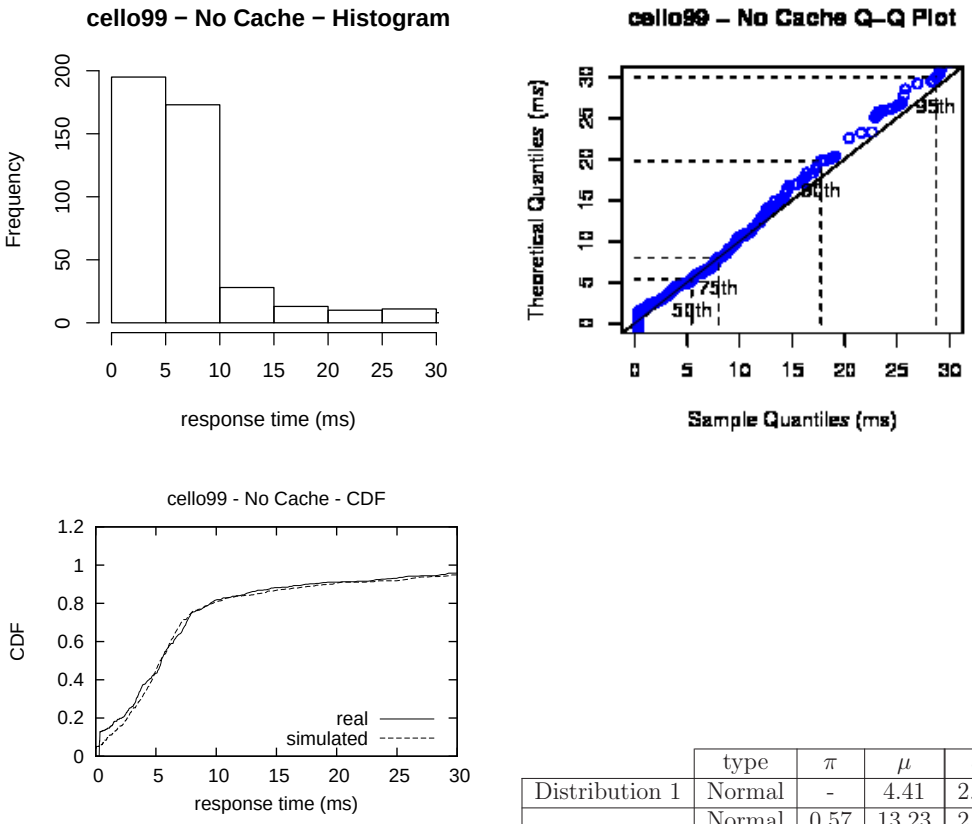


	type	π	μ	σ
Distribution 1	Normal	0.89	3.73	1.96
	Normal	0.11	10.82	5.71
Distribution 2	Normal	0.92	47.04	6.38
	Normal	0.08	29.42	3.99

Figure 5.7: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a Financial workload. The table shows the parameters of the modeled distributions. Caching is removed from the disk drive

Figure 5.7 shows the histogram, Q-Q and CDF's plots, for the Financial workload. In this trace, we identified two distributions, and both are mixtures. Their parameters are shown in the enclosed table. The mean size of the request is around 7 blocks. The trace is not bursty, and the enqueing time almost does not have influence in the response times. Also, the trace is very sequential.

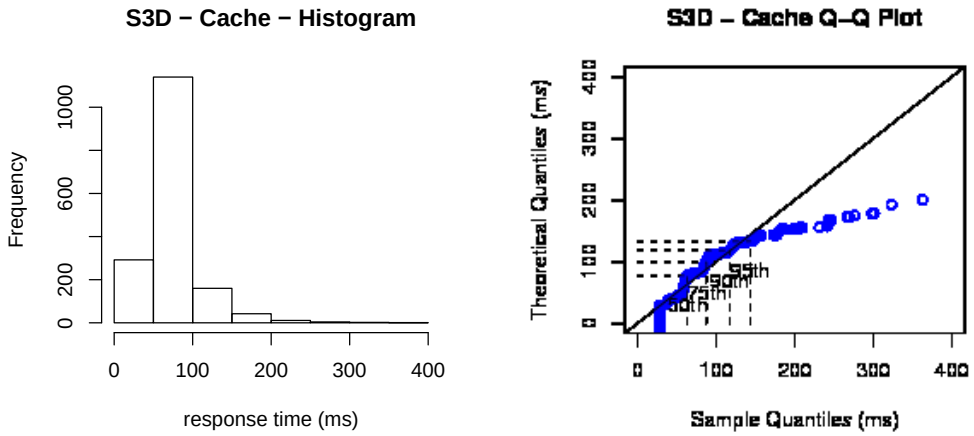
We identified three reasons for the two discovered distributions. When requests are sequential and there are no previous requests waiting to be serviced, response times are generated from part 1 of Distribution 1. When there are previous requests waiting to be serviced, response times are generated from part 2 of Distribution 1. Also, when a request arrives after a period of idleness, depending on the length of the period, response times may be generated from Distribution 2 or from part 2 of the Distribution 1. Response times from part 2 of Distribution 1 are generated when inactivity periods are longer than 1 second. Response times from Distribution 2 are generated when inactivity periods are around half a second. The goodness-of-fit of the model, for a relative mean error of 7.4%, can be seen in the shown Q-Q and CDF's plots.

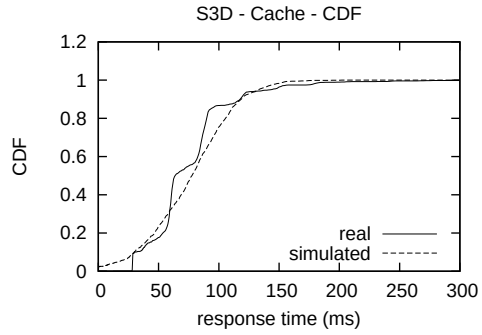


	type	π	μ	σ
Distribution 1	Normal	-	4.41	2.76
Distribution 2	Normal	0.57	13.23	2.77
	Normal	0.43	24.22	5.08

Figure 5.8: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a cello99 workload. The table shows the parameters of the modeled distributions. Caching is not activated in the disk drive

Figure 5.8 shows the histogram, Q-Q and CDF's plots, for the cello99 workload. In this trace, the mean request size is around 16 blocks. We identified two distributions, of which one them is a mixture, but not the other. Their parameters are shown in the enclosed table. Periods of burstiness alternate with periods of idleness. Periods of idleness are not very long, so they do not make longer response times, they just not produce longer response times as a result of enqueuing times. We identified two reasons for the two discovered distributions. When a request arrives after a period of idleness, and it is considered as sequential, its response time is generated from Distribution 1. On the other hand, the response time is generated from Distribution 2. The cache was deactivated. The model presents a relative mean error of 9%. Its goodness-of-fit can be visually appreciated in the Q-Q and CDF's plots.

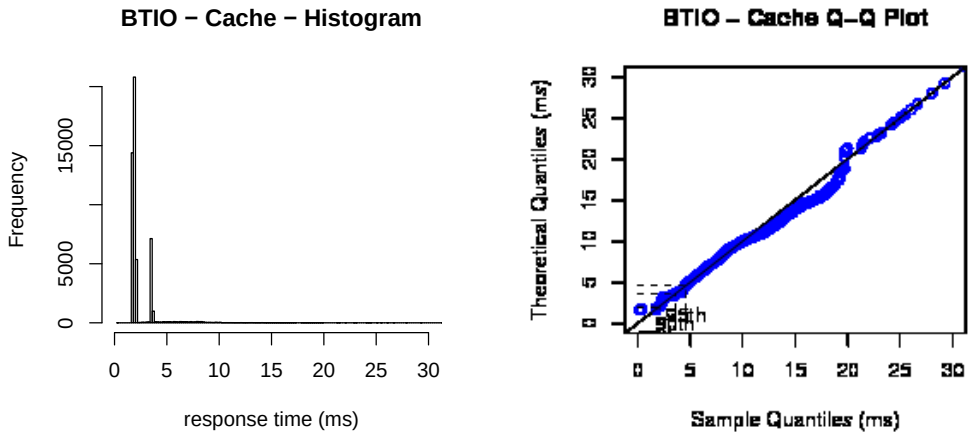




	type	μ	σ
Distribution 1	Normal	74.9	35.8

Figure 5.9: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a S3D workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive

Figure 5.9 shows the histogram, Q-Q and CDF's plots, for the S3D workload. In this case, the disk cache was activated. As when the cache was deactivated, we identified only one distribution. The size of each request is the same (2048 blocks), and here, the enqueuing times keep dominating the response times. Just the usage of the write caching makes response times a little shorter, but enqueuing times are still the dominant factor. That is because of the burstiness of the trace, which cannot be completely avoided, despite of the usage of the write buffers. In this trace, as we identified only one distribution we did not have to distinguish among several distributions and therefore, neither the reasons for them. All the response times will be generated from the same distribution. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 20%. Here, the disparity in the CDFs is due to the error when fitting the distribution.

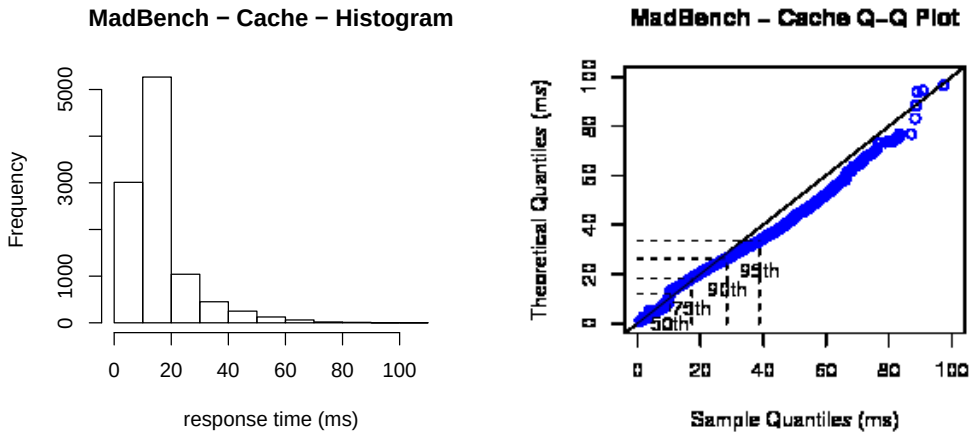


	type	π	μ	σ
Distribution 1	Normal	0.83	1.93	0.06
	Normal	0.17	3.54	0.11
Distribution 2	LogNormal	-	1.97	0.33

Figure 5.10: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a BTIO workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive

Figure 5.10 shows the histogram, Q-Q and CDF's plots, for the BTIO workload. Here, the disk platters are working altogether with a disk cache. We identified three distributions, whose parameters are shown in the enclosed table. Part 1 of Distribution 1 generates response times when writes are recorded in the cache, and the requested reads have been previously anticipated. As can be seen in the histogram, there is a very high probability that this happens. That is because the trace is very sequential and also not very bursty. Sequentiality benefits the fact that some data that will be requested soon, and have not been requested yet, can be anticipated. When a trace is not bursty, enqueuing times do not affect either subsequent response times.

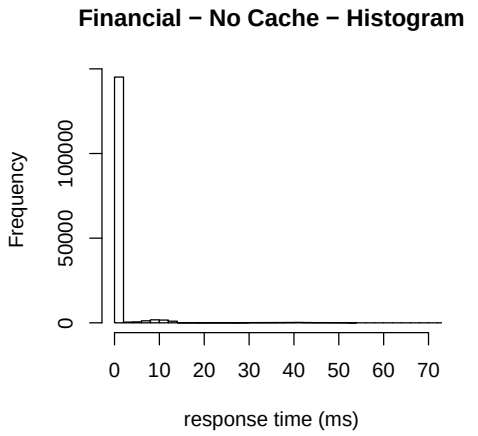
Response times from Part 2 of Distribution 1 are generated when a request arrives, and the previous request has not been serviced yet. Response times from Distribution 2 are generated when the current request is serviced by the platters and not by the disk cache. The trace presents some periods of idleness, and when a requests arrives after them, response times are generated from Distribution 2. The goodness-of-fit of the model is visible in Q-Q and CDF's plots. The relative mean error is 4.9%.

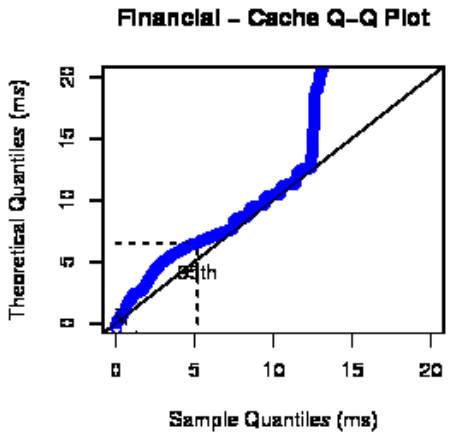


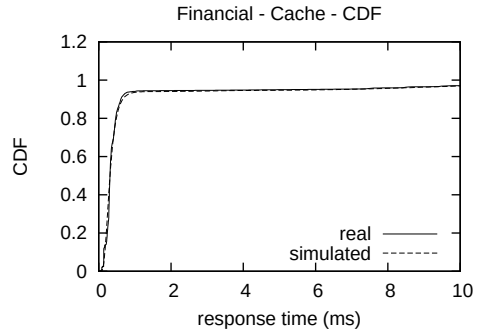
	type	μ	σ
Distribution 1	LogNormal	2.48	0.63

Figure 5.11: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a MadBench workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive

Figure 5.11 shows the histogram, Q-Q and CDF's plots, for the MadBench workload. The usage of the disk cache was activated. We identified one distribution for this trace. As a result of this, we did not either have to distinguish among different distributions and neither finding their reasons out. Here, all the response times are generated from the same distribution. Due to the high burtiness of the trace, and despite of the use of the cache, here again the dominant factor in the response time, is the enqueuing time. As was previously said, the trace is not very sequential, making useless the read-ahead effect during the read periods. The distribution is slightly different from the previously modeled distribution, when the cache was deactivated, because of the immediate reporting during the write periods. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 14.3%. Here, the disparity in the CDFs is due to the error when fitting the distribution.





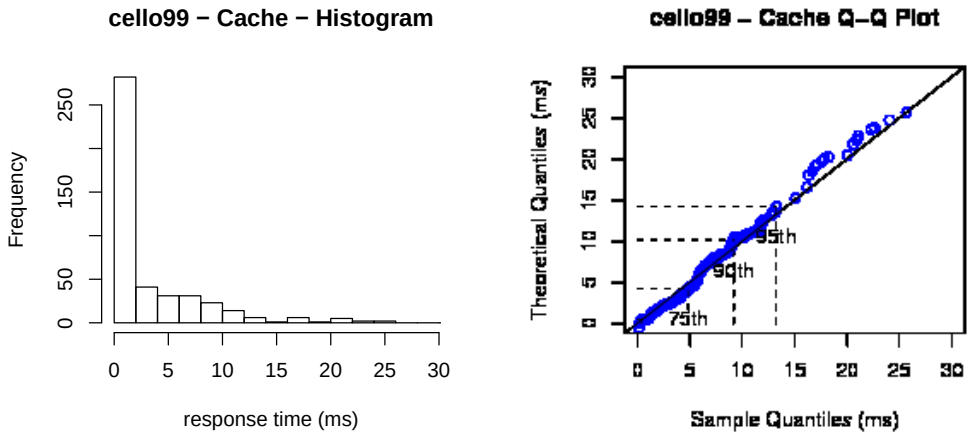


	type	π	μ	σ
Distribution 1	LogNormal	-	1.11	0.46
Distribution 2	Normal	0.88	9.63	2.48
	Normal	0.12	2.96	0.76
Distribution 3	Normal	0.91	39.89	6.41
	Normal	0.09	19.83	3.19

Figure 5.12: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a Financial workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive

Figure 5.12 shows the histogram, Q-Q and CDF's plots, for the Financial workload. In this case, the usage of the disk cache was activated. As in the case without the cache, we identified two mixtures, which represent sequential accesses to the disk platters (part 1 of Distribution 2), influence of queuing times (part 2 of Distribution 2), and different lengths of idle periods (part 2 of Distribution 2 and Distribution 3). When the disk has been idle for more than 520 ms and less than 700 ms, response times are generated from Distribution 3. When the disk has been idle for more than 1000 ms, response times are generated from part 2 of Distribution 2. Also, we identified another distribution (Distribution 1) which generate response times from hits in the disk cache.

As we previously said, the trace is very sequential and has a big percentage of write requests, which makes it optimal for its usage with a cache. That is why, in the histogram, a great percentage of response times come from Distribution 1. The goodness-of-fit of the model is shown in Q-Q and CDF's plots. Relative mean error is 10.20%.



	type	π	μ	σ
Distribution 1	Normal	-	3.30	1.12
Distribution 2	Normal	0.58	11.85	2
	Normal	0.42	19.41	3.27
Distribution 3	Normal	0.64	4.43	2.63
	Normal	0.36	2.15	0.88

Figure 5.13: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a cello99 workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive

Figure 5.13 shows the histogram, Q-Q and CDF's plots, for the cello99 workload. In this test, the disk cache was activated. We identified three distributions, and two of them were mixtures. Response times from Distribution 1 model the immediate reporting from write accesses to the write cache. They also model the read hits on the read cache, which have been previously anticipated. The other two distributions model accesses to the platters, which cannot be serviced by the disk buffer. Distribution 3 models response times that result from sequential accesses, while distribution 2 models response times that come from non sequential accesses. Distribution 2 also models response times from requests that have to wait until the previous ones have finished. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 12%.

Figures 5.14 and 5.15 show the CDF results of executing the five previously mentioned traces on our analytical variate generator (bbm), DiskSim, and the Seagate Cheetah 10K.7 disk.

We got the DiskSim model parameters by running *DIXtrac* [SG00, SGLG02, BS02] on the Seagate Cheetah 10K.7 disk. We both plotted the CDF curves for the real disk and models outputs, and used the the demerit [RW94] error as our metric to validate both models against the real disk. We presented the demerit figures in relative terms, as a percentage of the mean response time.

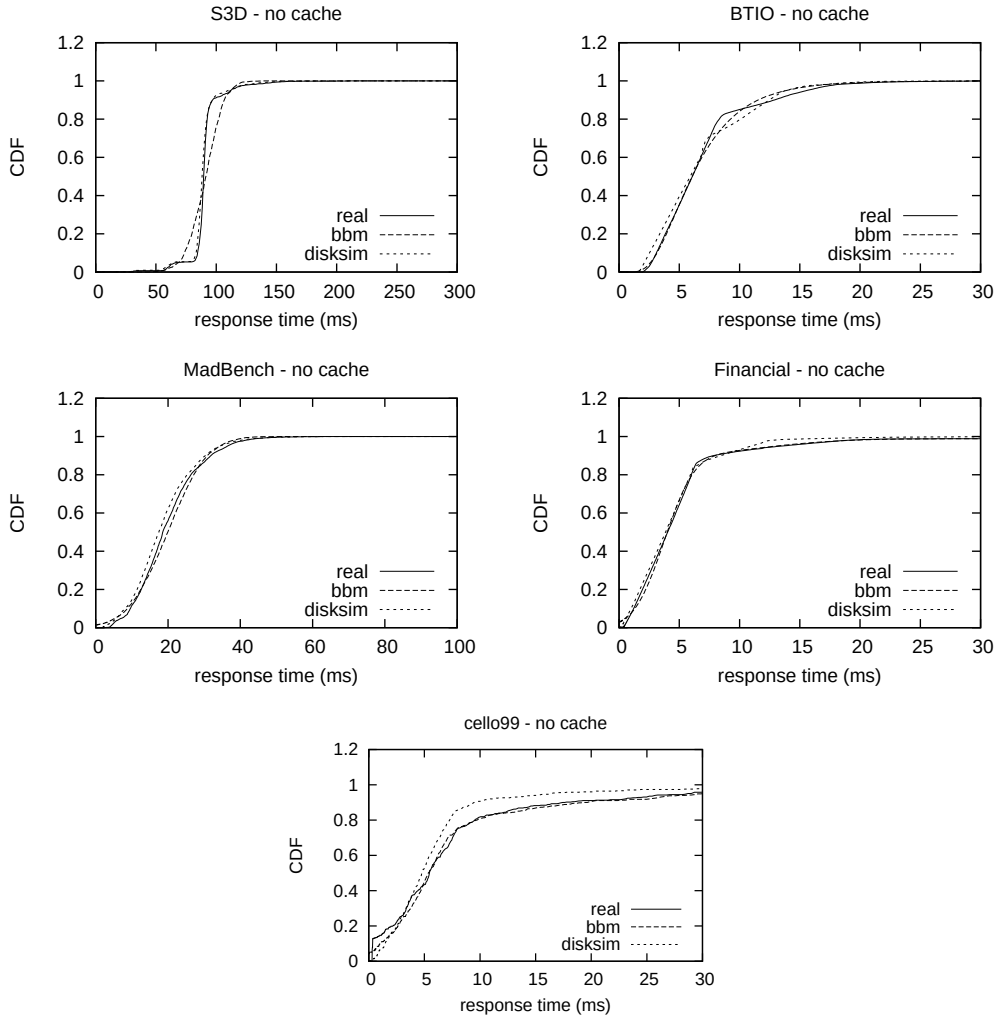


Figure 5.14: CDF's for response times from a Seagate Cheetah 10K.7 disk and several block traces. Caching is removed from the disk drive

Figure 5.14 shows the visual CDFs comparison of the five traces runs, on a disk with no buffer cache. Demerits of those distributions are shown in Table 5.2. Note that for some traces demerit figures are better in our approach than in DiskSim, and vice versa. In any case, most of them, keep roughly in the same range. The highest demerits turn out to be in the DiskSim case, for Financial (54%) and Cello99 (45%). In both cases, DiskSim did not

take into account effects on response times, when idle periods are around half a second. Other traces, like S3D and MadBench produced higher demerits in bbm (8% and 8.40%, respectively) than in DiskSim (5.6% and 6.70%, respectively), due to errors when fitting the distributions.

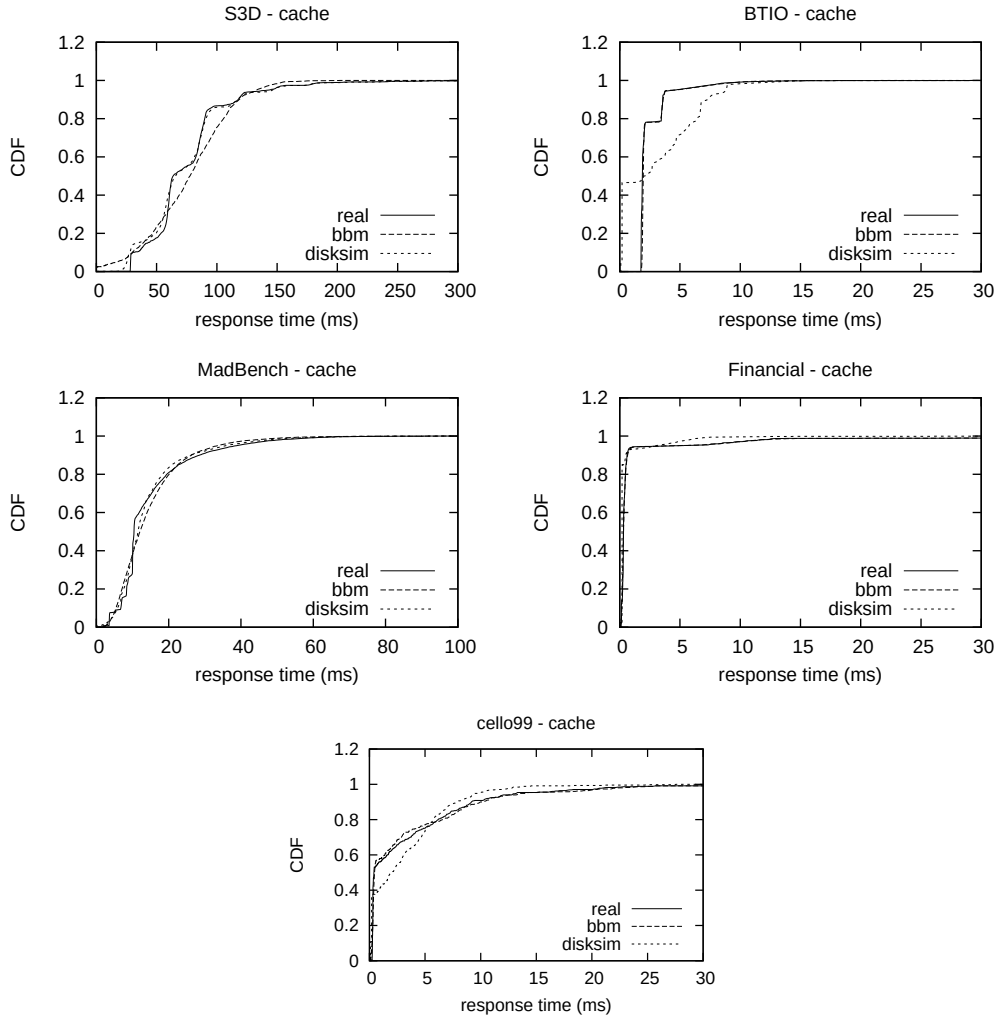


Figure 5.15: CDF's for response times from a Seagate Cheetah 10K.7 disk and several block traces. Caching is added to the disk drive.

Figure 5.15 shows the CDFs comparison of the five traces executions, on a disk that uses read-ahead an immediate reporting. Demerits of those distributions are shown in Table 5.2. Like in the non-cache approach, demerits are better for some traces in the bbm approach than in DiskSim, and vice versa. However, in the DiskSim approach demerits are out of range for some traces (BTIO, Financial, and Cello99). We think this is because *DIXtrac* did not perfectly identified the caching parameters or policies, for the Seagate Cheetah 10K.7 disk. However, other traces, like S3D and MadBench produced higher demerits in bbm (20% and 14.30%, respectively) than in DiskSim (4.70% and 8.70%, respectively). That is because of errors produced when fitting distributions. In the DiskSim cases, caching

did not affect response times for S3D and MadBench because both of them are very bursty and also their requests sizes are big. This makes enqueuing times to outshine possible hits on disk cache, and also demerits to be lower.

	Cache		No Cache	
	Disksim	bbm	Disksim	bbm
S3D	4.7%	20%	5.6%	8.0%
BTIO	94%	4.9%	13%	8.3%
MadBench	8.70%	14.30%	6.70%	8.40%
Financial	286%	10.20%	54%	7.40%
Cello99	51%	12%	45%	9%

Table 5.2: Demerits in relative terms for our black box model approach (bbm) and DiskSim

Figure 5.16 shows a summary of the demerit figures for our black box model approach (bbm), the one on the left, and for DiskSim on the right. Both graphs compare demerits when the disk uses the buffer cache and when does not. In both approaches most of the traces present demerits lower than 20%. In the DiskSim approach (right), the three peaks previously commented from BTIO (94%), Financial (286%), and Cello99 (51%) appear for the cache approaches. Also, two peaks for Financial (54%) and Cello99 (45%), appear for the non-cache approaches, because in DiskSim, long idle times do not have effects in subsequent response times.

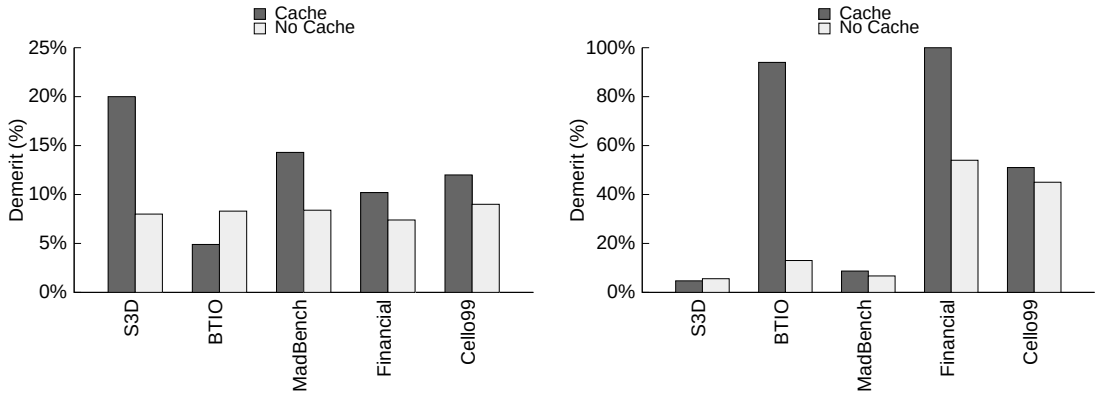


Figure 5.16: Demerits in relative terms for the black box model approach (left) and DiskSim (right)

Figure 5.17 shows how much our black box model approach is faster than DiskSim, in terms of speedup. Speedups are compared when the disk buffer cache is activated and when it is not. The execution times for the S3D trace are almost 600 times faster than for DiskSim. This is primarily due to the number of blocks that are demanded in the same request, which are higher than in any other trace. The other traces are still faster in the black box model executions, which translates into the fact that our black box model is highly efficient.

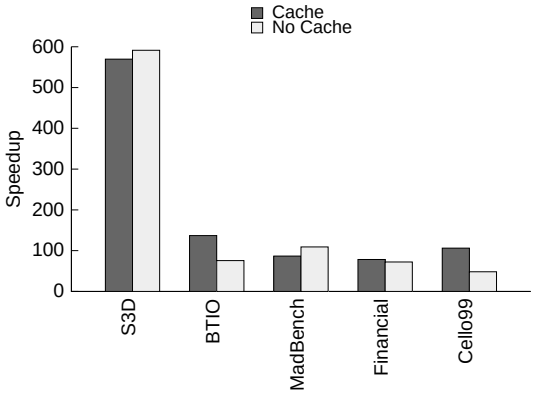


Figure 5.17: Speedups figures. They represent how much the black box model approach is faster than DiskSim.

5.1.3 Testing models based on several real traces

To demonstrate the feasibility of the model, in this section we test several non-trained traces, by using the previously constructed black box model. For non-trained traces we mean traces that have not been previously used, in the construction of the models. We picked several block I/O traces from the SNIA IOTTA repository [sni11]:

- *WebResearch* represents a web-based management of several projects, using the Apache web server [apa11].
- *Online* represents a course management system of a department, using Moodle [moo11].
- *WebUsers* represents a web server hosting faculty, staff and graduate students web sites.
- *WebMail* represents the web interface to the department web server.

Figures 5.18, 5.19, 5.20, and 5.21 show the Q-Q and CDF results of executing the four previously described traces on our variate generator (bbm) and the Seagate Cheetah 10K.7 disk.

We both plotted the Q-Q and CDF curves for the real disk and for the model, and used the demerit [RW94] error as our metric to validate our model, against the real disk. We presented the demerit figures in relative terms (as a percentage of the mean response time).

In this case, our black box model (bbm) is based on the five real traces described in the previous section. As was previously said, to predict response times by using our black box model (bbm), one of the modeled traces is chosen. Our model chooses the one which has similar characteristics of size, queuing times and sequentiality. For all the four traces, that trace is Financial. That is because, in all of them, request sizes are mostly 8 blocks, accessed LBNs are very sequential, and the traces are not very bursty. As a result of this last characteristic, queuing times almost do not exist.

Figures 5.18 and 5.19 depict the visual Q-Q and CDF comparisons of the four traces runs, by deactivating the buffer cache on the disk. Demerits of those distributions are shown in table 5.3. For the four traces, predictions are made from the model constructed in figure 5.7, which is the constructed one for the Financial trace. So, for the four traces, response times are generated from the distributions shown in the enclosed table.

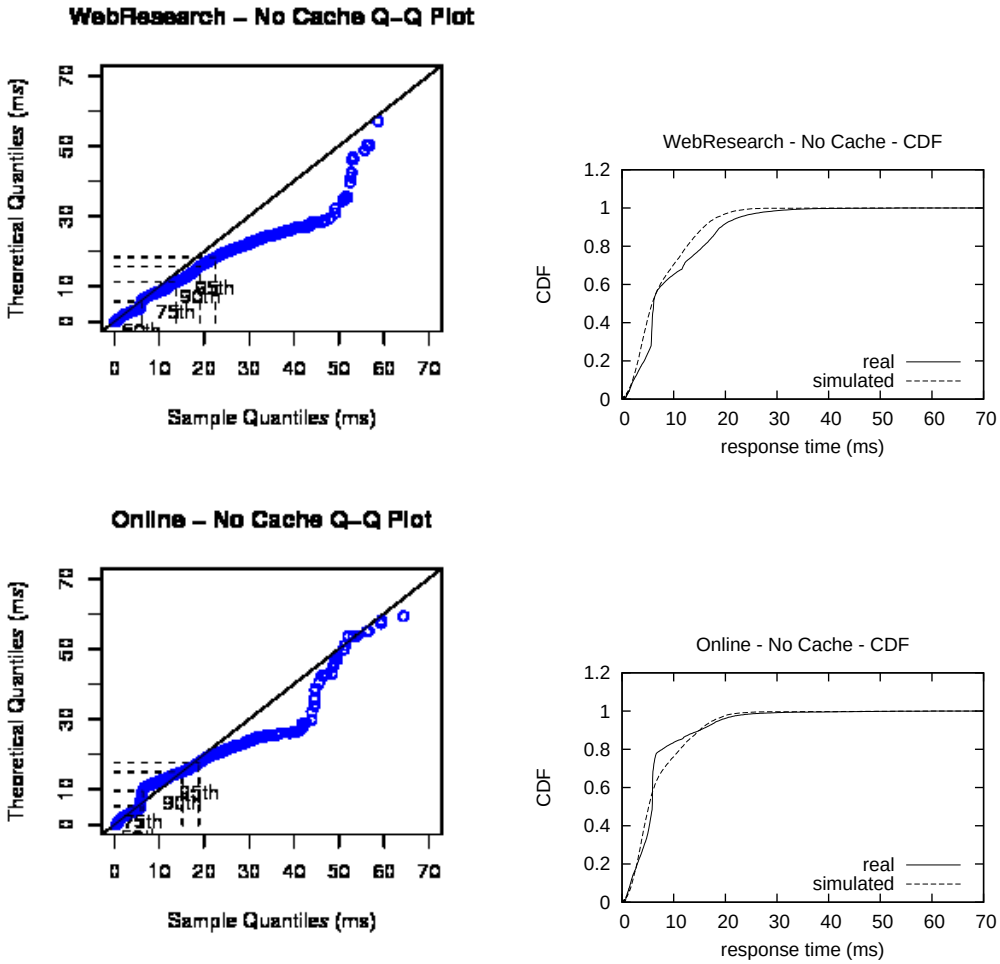


Figure 5.18: CDFs and QQ plots for response times from our black box model (bbm) approach. Non trained traces are executed. Caching is not activated in the disk drive.

Both WebResearch and Online (figure 5.18) present the worst results of the four traces. The errors are due to a main cause: Queuing times. For the Financial trace, there are almost not queuing times, and when they appear, they can be simulated by generating response times from part 2 of Distribution 1.

Both for WebResearch and Online, queuing times are more common than in Financial, and real response times turn out to be longer than the ones generated from part 2 of Distribution 1. That happens because if the current request has been long, its length does not affect the subsequent requests. That is because generation of response times are independent, and if the current response time was long, the subsequent request does not

have to be also long. In the specific case of the Online trace, many of the queuing effects produce response times that should be shorter than the values generated by part 2 of Distribution 1.

Also, both for WebResearch and Online, response times in the range that belongs to part 1 of Distribution 1 are shorter than real response times, for the same range. This is because, when a current request is enqueued, and it must wait for less than 4 seconds, its response time is generated from part 1 of Distribution 1, being, again independent of the previous requests. This means that the current response time could be shorter than the real enqueing time, as it happens.

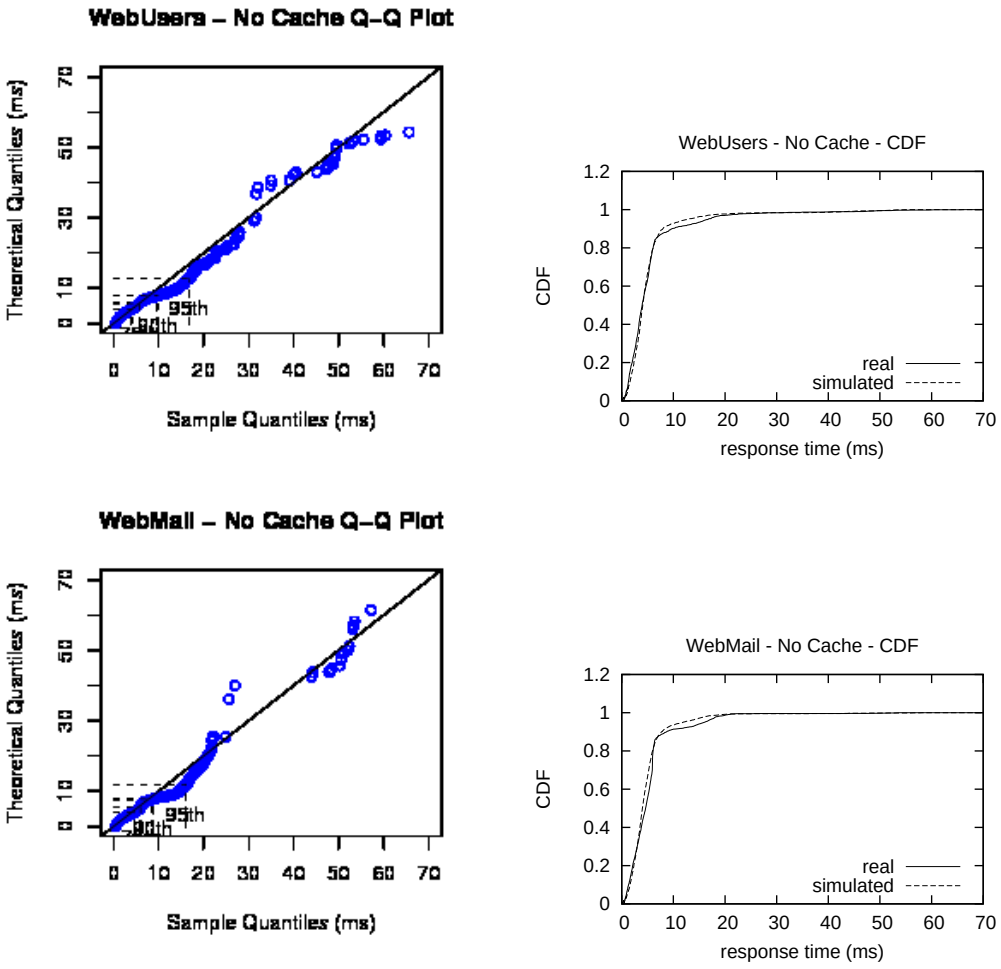


Figure 5.19: CDFs and QQ plots for response times from our black box model (bbm) approach. Non trained traces are executed. Caching is not activated in the disk drive.

That difference between the real and simulated distributions is not so noticeable in WebUsers and WebMail cases (figure 5.19), where queuing times are not so common, but are still common, specially after servicing a request, when the disk has been idle for more than a second.

Figures 5.20 and 5.21 depict the visual Q-Q and CDF comparisons of the four traces runs on a disk with buffer cache. Demerits of those distributions are also shown in table 5.3. For the four traces, predictions are made from the model constructed in figure 5.12. Here, demerits are a little higher. That is because of the use of the cache, which produces shorter response times, and the mean is also shorter. Calculating the relative error involves dividing by that mean, and no matter how short the differences between the CDFs are, when a different is too long that affects the relative error. Here, also the errors are due to the same cause: Queuing times. The effect of them is more noticeable after servicing a request, when the disk has been idle for more than a second.

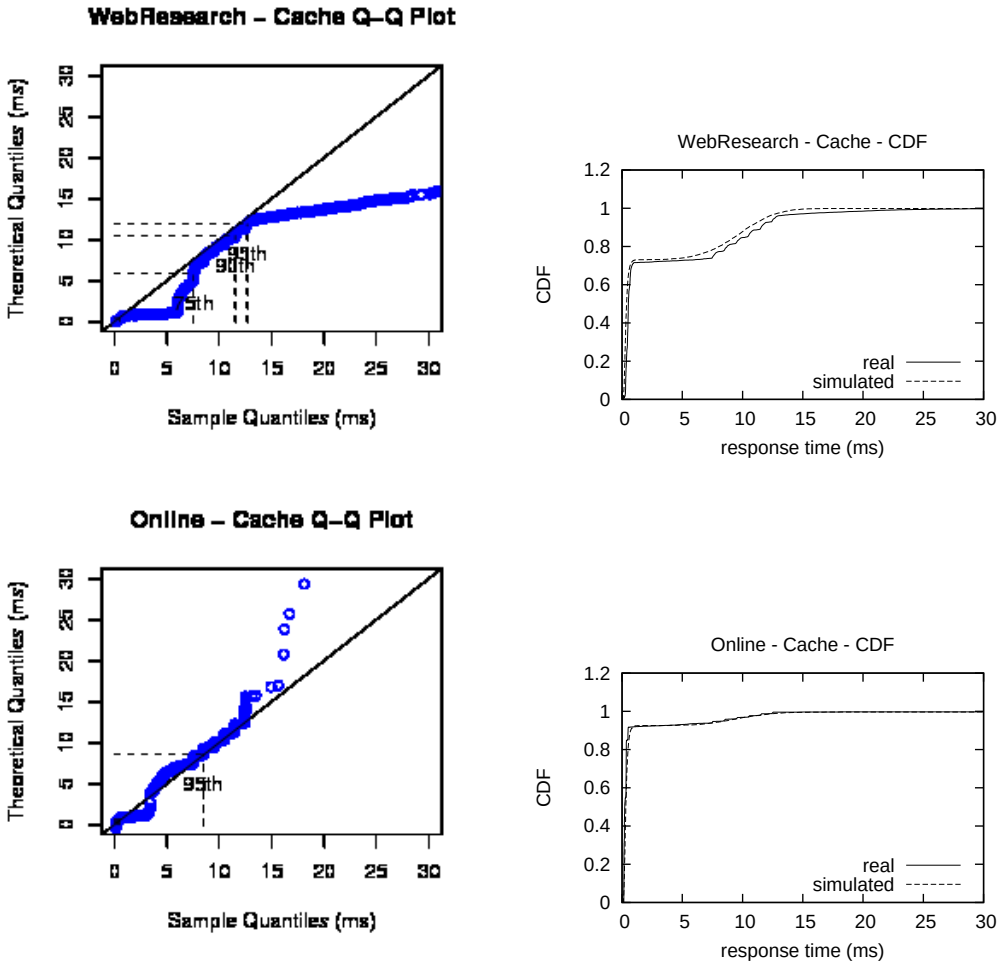


Figure 5.20: CDFs and QQ plots for response times from our black box model (bbm) approach. Non trained traces are executed. Caching is active in the disk drive.

As shown in table 5.3, the lowest demerit, when caching is active, is obtained for Webmail (25%). That is because response times range covers higher values, and also big differences between real and simulated values are not very often. Next demerit is obtained for WebResearch and Online (31%). In WebResearch, response times range covers the highest values, making differences between real and simulated response times not to have

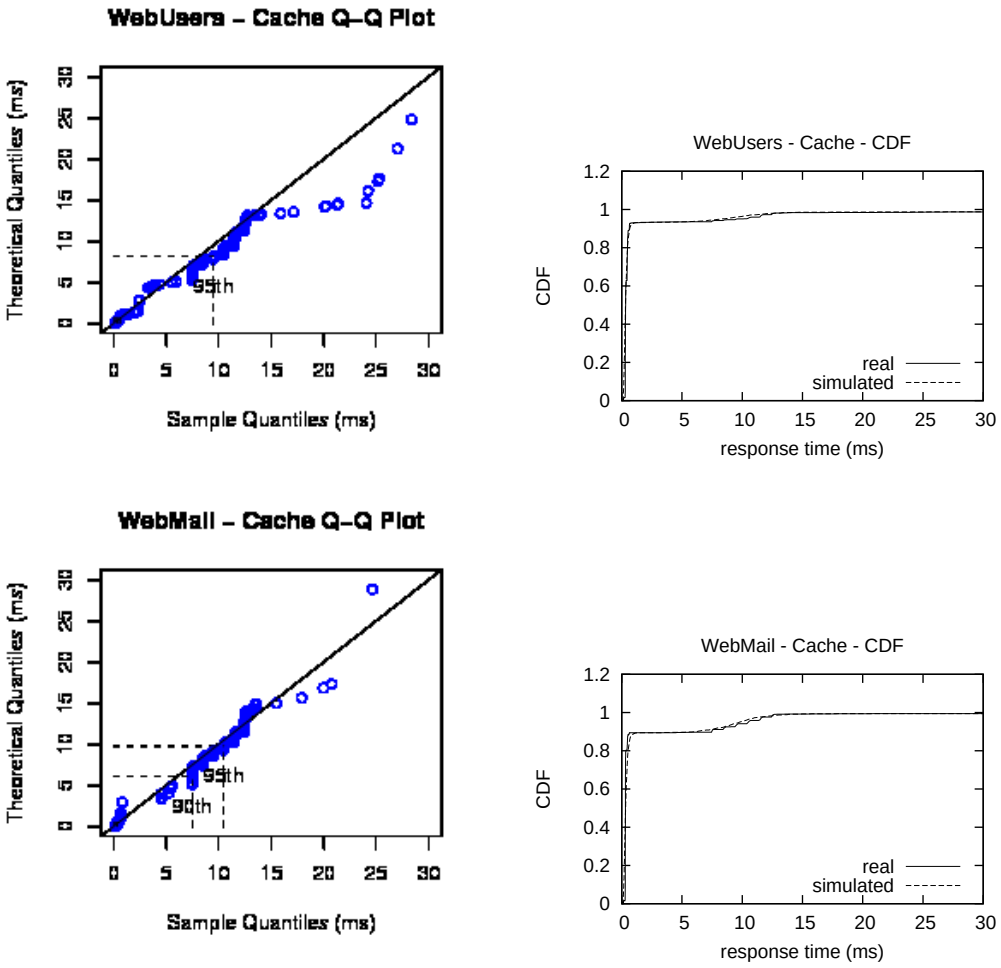


Figure 5.21: CDFs and QQ plots for response times from our black box model (bbm) approach. Non trained traces are executed. Caching is active in the disk drive.

too much effect on the demerit calculation. In Online and WebUsers, response times range does not cover high values, but the few differences between real and simulated response times have effect on de demerit calculation, specially for WebUsers (39%).

	Cache	No Cache
WebResearch	31%	24%
Online	31%	25%
WebUsers	39%	21%
Webmail	25%	21%

Table 5.3: Demerits in relative terms for our black box model approach (bbm). Non trained traces are executed.

As a future work, we plan on learning from tested new traces, in order to consider new characteristics that have not been covered by the previously trained traces.

5.1.4 Testing models based on a synthetic trace

In this section, we construct, and test, a model based on a synthetic trace. As was previously said, to construct a model based on a synthetic trace, we first obtain the experimental data from our real disk. When constructing the model, we do not take into account queuing times. So, when obtaining the experimental data, we restrict the maximum number of pending I/O requests, that can be enqueued in the disk to 1. We do not take into account queuing times, when constructing our model, because in our model, queuing times are modeled on the fly, making it more general and versatile.

For the synthetic trace, among different available options, we chose the SPC Benchmark v1.10.1 [Cou06] workload defined by the Storage Performance Council [spc11]. As it is said in [GM05], the traces generated by the benchmark simulate real world environments, that are typical for business typical applications like OLTP systems, database systems and mail server applications.

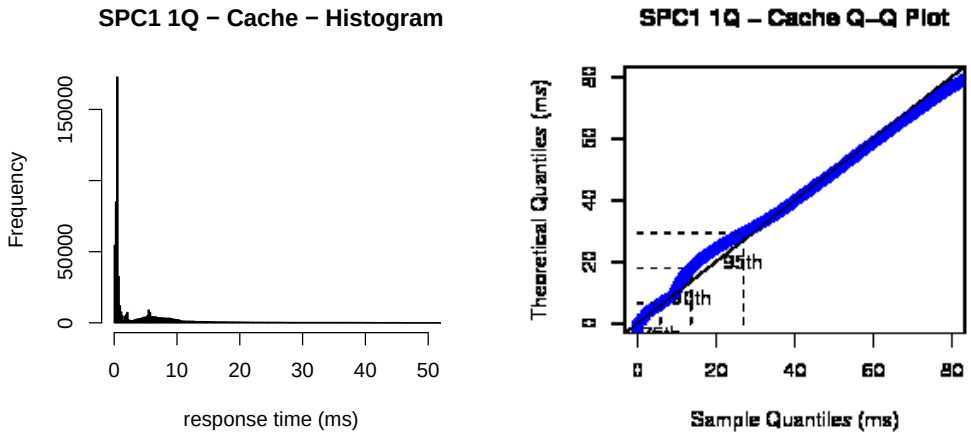
When constructing the model, we plot the histogram for the data obtained by using *play* (our service time measurement program). By representing the histogram, we identified the possible distributions to fit to. After that, we fit the previously identified distributions to theoretical distributions, and show them in a table. Then, we find the causes for them, and on the basis of those causes, we construct the model. We replay the synthetic trace again on the recently constructed model and show the Q-Q plot to judge the goodness-of-fit of the model to the experimental data, and the CDF to see how well they superimpose.

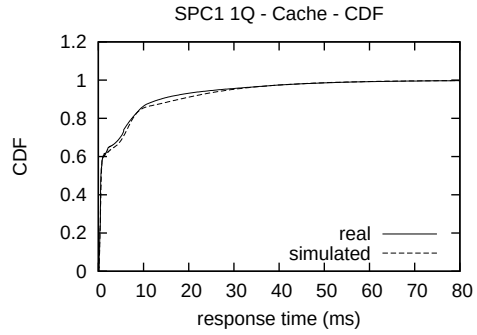
When testing the model, we replay several real traces also on the recently constructed model and show the Q-Q plots and CDFs.

For the real traces, we chose two of the bunch of previously described traces. We selected the ones that were more similar in characteristics to the SPC-1 workloads. Specifically, we selected the ones that were more similar in the range of request sizes to the SPC-1 workloads. Those selected traces were Financial [uma11], and Cello99 [Cel11]. As previously said, Financial is the I/O core of an OLTP application gathered at a huge financial organization. On the other hand, Cello99 is a shared compute/mail server from HP Labs. To see the effect of the online calculation of enqueuing times, we also used another trace, WebSearch [uma11]. It is a famous search engine server, which executes about 4 million read requests over 6 disks, only during 4 hours.

For simplicity, we did not deactivated the disk buffers. We just both construct the model and predict from it, by using the read-ahead and immediate reporting by default.

Figure 5.22 shows the histogram, Q-Q and CDF's plots, for the SPC-1 workload, and the Seagate Cheetah 10K.7 disk. As we previously said, the usage of the disk cache was activated. We identified three distributions, whose parameters are shown in the enclosed table. Response times from Distribution 1 - part 1, model the immediate reporting from write accesses to the write cache. They also model read hits on the read cache, which have been previously anticipated. The other two distributions model accesses to the platters, which cannot be serviced by the disk cache. Distribution 2 models response times that result from sequential accesses, while Distribution 3 models response times that come from non sequential accesses. Distribution 2 also models response times that result from write requests with sizes bigger than 64 blocks. As we previously said, enqueuing times are not





	type	π	μ	σ
Distribution 1	Normal	0.91	0.45	0.20
	Normal	0.09	1.89	0.84
Distribution 2	Normal	-	6.42	1.79
Distribution 3	LogNormal	-	3.11	0.60

Figure 5.22: Histogram, Q-Q and CDF's plots for response times from a Seagate Cheetah 10K.7 disk and a SPC-1 workload. The table shows the parameters of the modeled distributions. Caching is active in the disk drive. Queuing times are not considered in the model.

included in the identified distributions. Both the Q-Q and CDF's plots show the goodness-of-fit of the model, and the relative mean error is 25.5%.

Figure 5.23 depicts the visual Q-Q and CDF comparisons of the four traces runs, by using the model constructed with the synthetic trace. Demerits of those distributions are shown in table 5.4.

For the four traces, predictions are made from the model constructed in figure 5.22. When predicting, the four traces calculate queuing times on the fly, by checking if the previous requests have finished or not.

The SPC1 trace is the same trace as SPC1 1Q. The difference between them is that, when measuring service times, in SPC1 1Q the maximum number of enqueued requests is limited to 1, and that does not happen with SPC1.

For all the traces, both the modeled and real distributions follow the same trend, although the made mistakes, when predicting, may mean shorter queuing times, as in SPC1 trace, or bigger queuing times, as in WebSearch, Financial, and Cello99 traces.

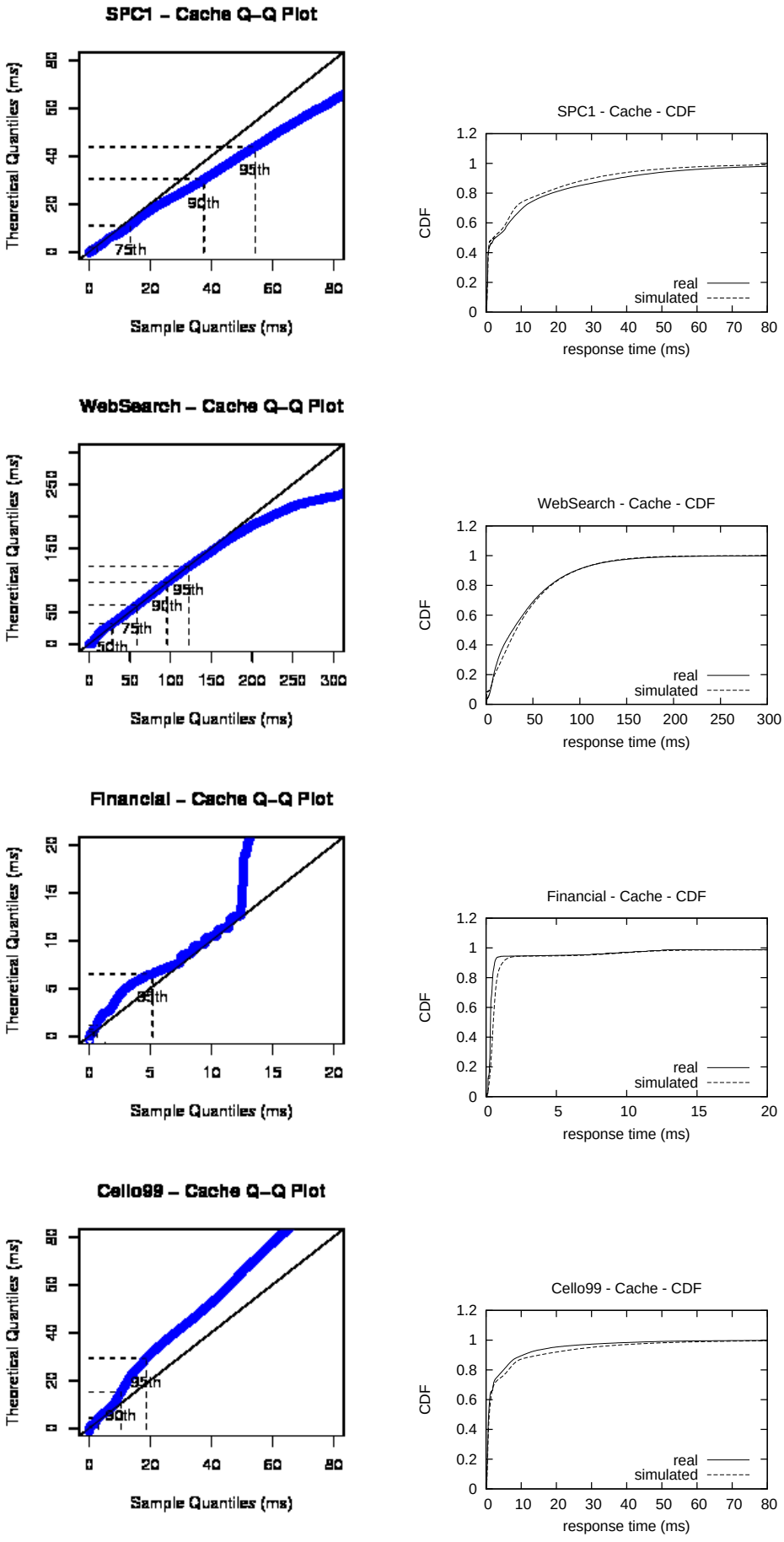


Figure 5.23: CDFs and QQ plots for response times from our black box model (bbm) approach. Non trained traces are executed. Caching is active in the disk drive.

In the specific case of Financial, although queuing times are not very common, response times from the model keep being bigger than from the real trace. That is because, in Financial, mean request sizes are smaller than in SPC1, which covers a wide range of sizes, generally bigger than in Financial.

Note that in Cello99, SPC1, and SPC1 1Q, response times fit worse when response times are bigger than 10 ms. Note also that the response times range of Distribution 3 starts from 10 ms. We think that is because Distribution 3 covers a wide range of response times, and in some cases it generates response times much bigger than it should, bringing on the mistakes in the predictions. We think that could be solved by splitting Distribution 3 on several distributions, but the tricky thing is identifying the causes for them.

Workload	Demerit
SPC1 1Q	25.5%
SPC1	59.6%
WebSearch	28.9%
Financial	48.3%
Cello99	87%

Table 5.4: Demerits in relative terms for our black box model approach (bbm). Non trained traces are executed.

5.1.5 Goodness-of-fit for individual response times

In order to show how simulated response times fit real response times for each individual request, we plotted two little samples in the course of two applications executions.

The first application is WebUsers, and it was executed on the disk described in Table 5.1, by activating its write-buffering and read-ahead mechanisms. WebUsers was also executed on DiskSim and on a black box model based on several real traces. In the specific case of WebUsers, the model chosen was the constructed one for the Financial trace.

The other application is Financial. It was also executed on the real disk, by activating its caching mechanisms. It was also executed on DiskSim and on a black box model based on a synthetic trace, SPC-1.

In order to calculate the goodness-of-fit for each individual request, a subtraction was done between the real response time and the simulated one. For less outcome, better the fit. Figures 5.25 and 5.24 show errors obtained for each individual request, in the course of the two application executions. Errors are shown for the black box models and for Disksim.

As can be seen, most simulated requests by using DiskSim obtain better fits than the ones obtained by the black box models. That is because DiskSim is very detailed and that makes it more accurate than black box models. Also, in black box models, when generating response times from a certain range, the values obtained are random, resulting different in some cases from the real ones.

However, for some requests in both applications, errors are higher by using DiskSim, than by using black box models, which is reflected on the CDF's superimposition plots.

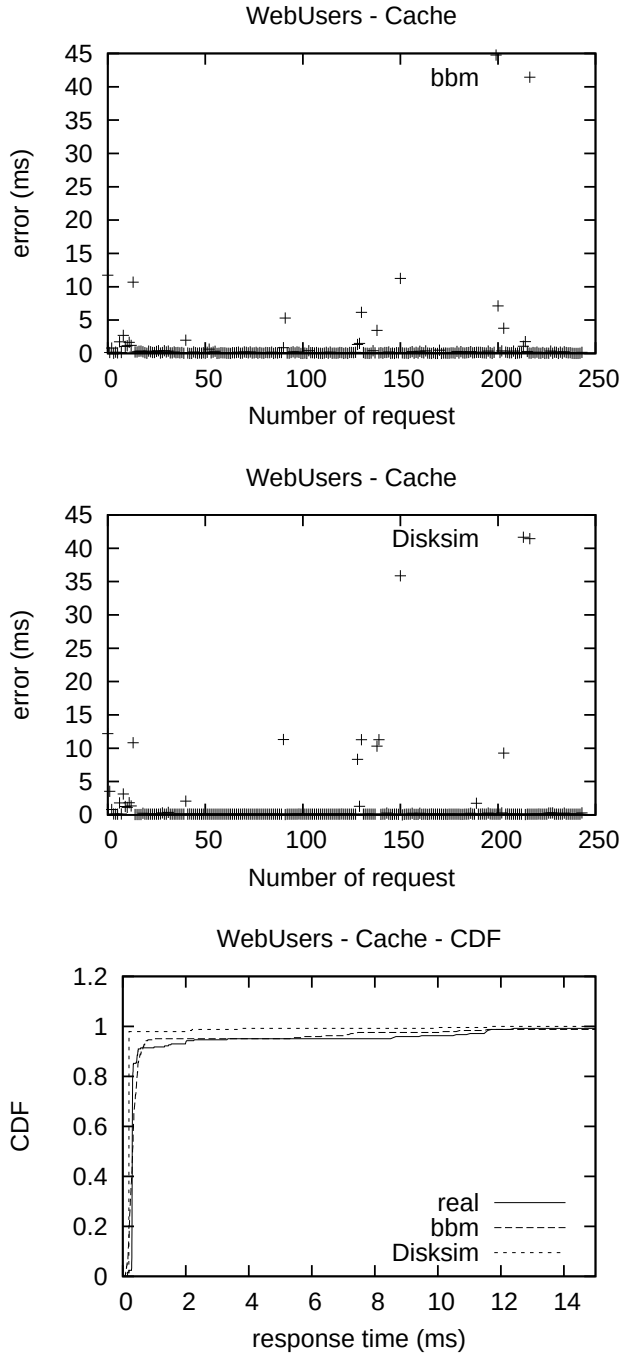


Figure 5.24: Goodness-of-fit for individual response times. Response times for a WebUsers trace are obtained by using a black box model and DiskSim. Caching is active in the disk drive.

That is because DiskSim does not consider effects from long inactivity periods, when simulating response times.

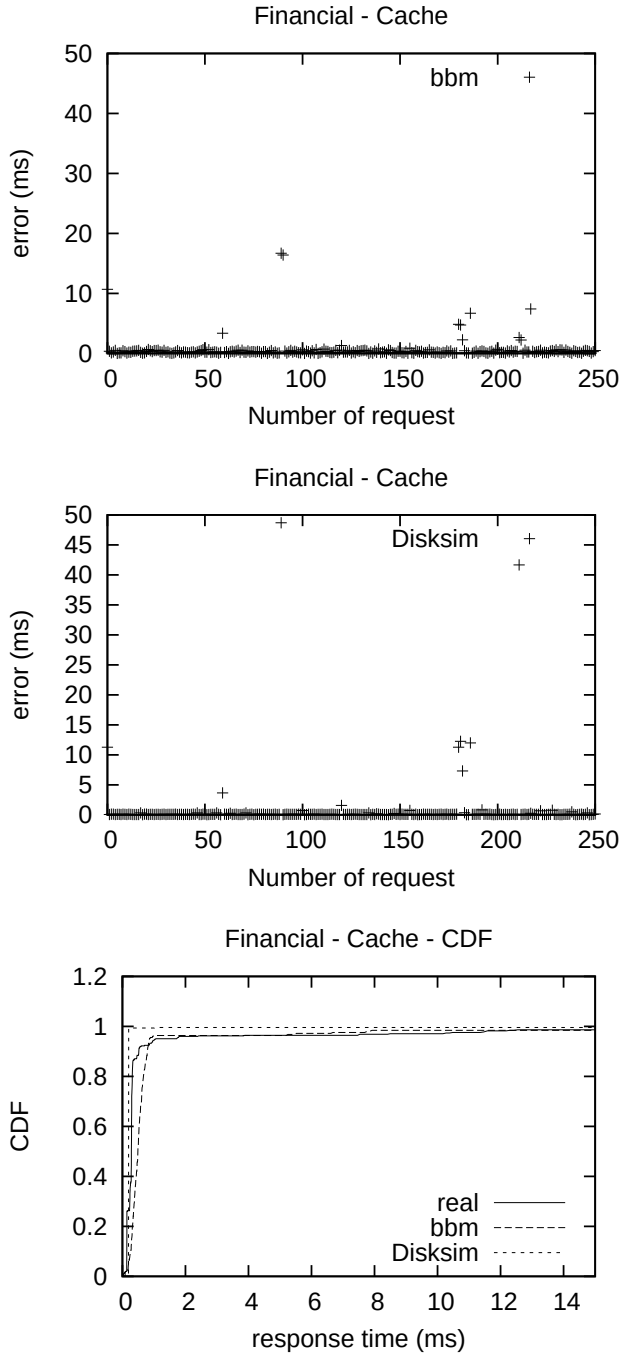


Figure 5.25: Goodness-of-fit for individual response times. Response times for a Financial trace are obtained by using a black box model and DiskSim. Caching is active in the disk drive.

5.2 Evaluation of Energy-aware Architectures

In this section, we evaluate the proposed approaches in Chapter 4. The section is organized in two subsections. First, in Subsection 5.2.1, we prove the feasibility of our write-buffering

policies in the SSD-based storage system. An analysis of monetary costs and improvements is provided. Also, performance evaluations are shown. Second, evaluation of several of the prefetching policies, is presented in Subsection 5.2.2. Analysis for both synthetic and real traces are presented. Both subsections apply the economic model described in Section 4.5

Table 5.5: Simulated disk and SSD specifications. The minus character means that specification is not applicable.

Specification	Value	
Model	Seagate Cheetah 15K.50	Samsung K9XXG08UXM
Type	Magnetic disk	SSD
Power consumption (idle)	12 W	0.5 W
Power consumption (active)	17 W	1 W
Power consumption (standby)	2.6 W	-
Duty Cycles	50,000	-
Erase Operations	-	100,000
Price per GB	0.14 $	9.2 $
Capacity	146 GB	2 GB - 16 GB

In order to evaluate our solution we have implemented a hybrid SSD-based architecture simulator in the widely used general purpose simulation framework OMNeT++ [omn12]. The simulator includes disk instances of one of the most representative disk simulators, namely DiskSim 4.0 [GWW+99]. SSD devices are simulated by instances of the SSD extension module inside DiskSim platform. As described in [APW+08] the model counts with several realistic SSDs device configurations. Hardware specifications used in this thesis for both magnetic and SSD devices are summarized in Table 5.5.

5.2.1 Write-buffering evaluation

As commented in Section 3.3.2, researchers have investigated several I/O intensive parallel scientific applications, such as MADBench2 [mad11], S3D [s3d11], and BTIO [bti11], to mention a few.

We have evaluated our architecture in terms of energy consumption and I/O performance, using real traces of the previous cited high performance applications. Traces configuration is summarized in Table 5.6. In case of S3D, we count with traces obtained from Red Storm [s3d11]. This system is a Cray XT3+ class machine, which uses Lustre [lus11] as file system. The block size used was 1MB and the storage system counted with 320 magnetic disks, in which file blocks are mapped round-robin over all disks. In case of

Table 5.6: Traces and systems configuration.

Specification	Trace		
Application	S3D	BTIO	MadBench
Compute nodes	6400	64	64
I/O nodes	320	4	4
Storage nodes	320	4	4
File system	Lustre	PVFS2	PVFS2
Block size	1024KB	64KB	64KB
File size	450GB	6.4GB	9.3GB
Architecture	Cray XT3+	Beowulf cluster	Beowulf cluster

BTIO and MadBench, the traces were obtained on a cluster with 4 I/O nodes, which uses PVFS [CCL+02] as parallel file system. The block size was set to 64 KB and the storage system counted with 4 magnetic disks, in which file blocks are mapped round-robin over all disks.

In order to prove the feasibility of our write-buffering policies in the SSD-based storage system, we have designed three different scenarios: isolate, working day, and concurrent workloads. In each case, we analyze the cost and the amortization in a seven year period, by using the economic model describe in Section 4.5. As was previously said, we chose that frame of time, considering that, typically, wear in disk drives is high after years 5-7 of operation under normal conditions [Sik07].

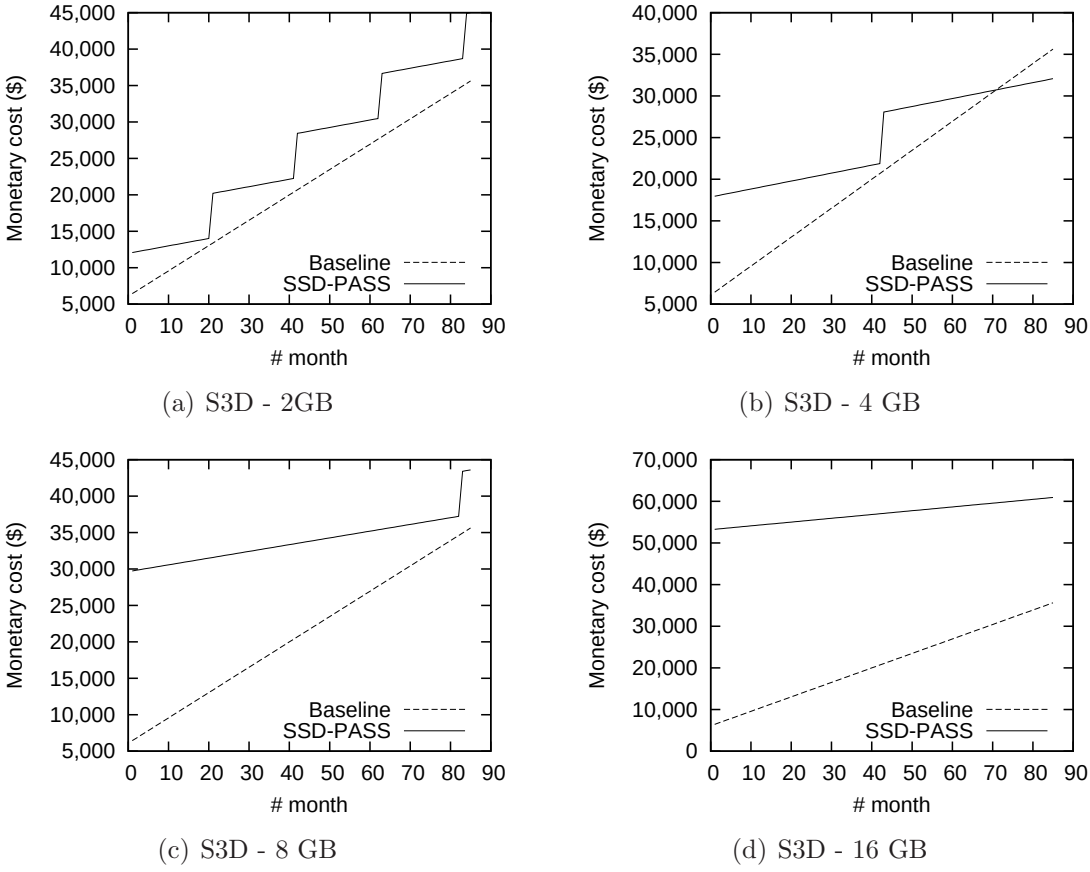
5.2.1.1 Isolated Workloads

In this first scenario, we execute each workload independently, in other words, every application uses the storage system in a dedicated way. This scenario is similar to modern supercomputers such as Blue Gene and Cray, in which applications run in a isolated partition (namely *pset* in case of Blue Gene supercomputers).

Figures 5.26, 5.27, and 5.28 show the total monetary cost in the course of a seven year period, when our power aware storage subsystem (SSD-PASS) is used and when it is not (Baseline). Figures represent monetary cost improvements, for traces S3D, BTIO, and MadBench, respectively. For every month, the cost of the energy consumed is reflected altogether with the cost of replacing devices with new ones. We consider that a replacement is demanded when the maximum number of erase operations per block is reached in a SSD, or when the number of duty cycles in a disk device is up to its maximum. The initial budget in our power aware approach goes over the initial budget in the Baseline approach. This is because in the Baseline approach the cost of acquisition comes from purchasing disks, what in the power aware one comes from buying both disks and SSDs. In the long run, that initial budget is paid for itself.

All the figures show amortization costs for 2 - 16 GB sized SSDs. As we will show, SSDs' sizes have a very important effect in the amortization results. Contrary to expectations, bigger sizes not always involve bigger monetary cost improvements. Having bigger SSDs'

sizes avoids disk spin-ups, and hence, disk replacements. However, as the price per Giga is still very high in SSDs, when SSDs' sizes are very big, it is not easy to amortize initial budgets, and getting monetary costs improvements.



(a) S3D - 2GB

(b) S3D - 4 GB

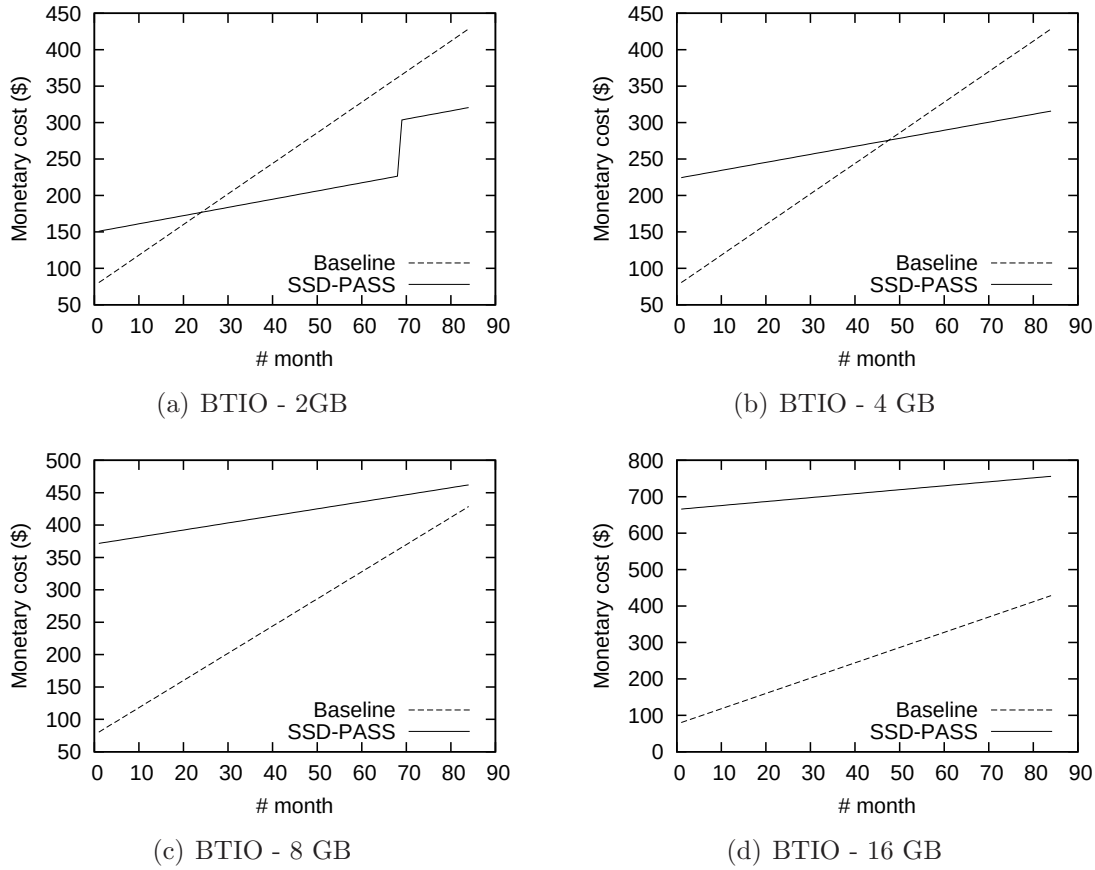
(c) S3D - 8 GB

(d) S3D - 16 GB

Figure 5.26: Monetary costs in Isolated Workloads and S3D.

Amortization costs for S3D are shown in Figure 5.26. Improvements are shown for SSDs sizes from 2 to 16 GB. Using SSDs of size 2 GB, amortization is never reached. That is because, every 20 months new disks are purchased, exceeding the budget of the baseline approach. That happens due to the small size of SSDs, which are filled more frequently, and also make disk drives start/stop more frequently. Using 4 GB sized SSDs, amortization is reached at almost half of the execution of the application, but the purchase of new disks at month 42 exceeds the budget of the baseline approach until month 71. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult to amortize. Using a SSD of size 4GB we obtain an energy consumption improvement of 9.36%.

Amortization costs for BTIO are shown in Figure 5.27. Improvements are shown for SSDs sizes from 2 to 16 GB. Using SSDs of size 2 GB, amortization is reached at month 27. Purchasing new disks at month 68 does not mean exceeding the budget of the baseline



(a) BTIO - 2GB

(b) BTIO - 4 GB

(c) BTIO - 8 GB

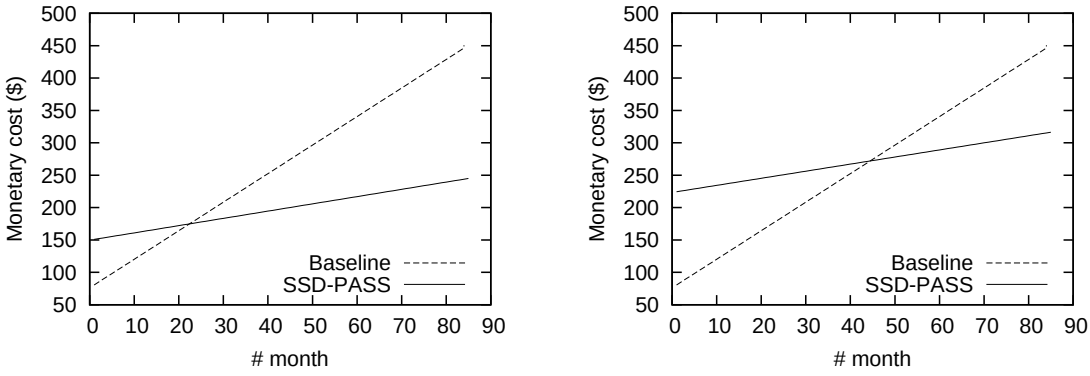
(d) BTIO - 16 GB

Figure 5.27: Monetary costs in Isolated Workloads and BTIO.

approach. From the amortization point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 4 GB sized SSDs, amortization is reached at month 48. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult to amortize. Using a SSD of size 4 GB we obtain an energy consumption improvement of 25.21%.

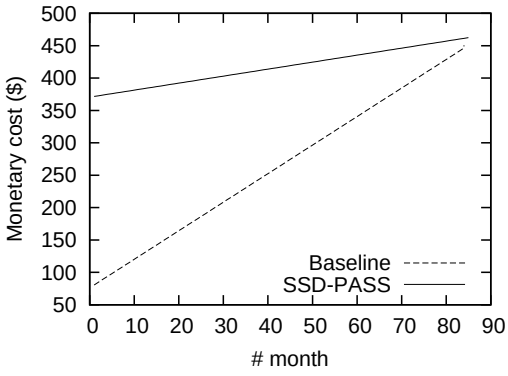
Amortization costs for Madbench are shown in Figure 5.28. Improvements are shown for SSDs sizes from 2 to 16 GB. Using SSDs of size 2 GB, amortization is reached at month 23. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 4 GB sized SSDs, amortization is reached at month 45. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult to amortize. Using a SSD of size 2 GB we obtain an energy consumption improvement of 66.504%.

The purchases of new disks for the S3D application are motivated by the number

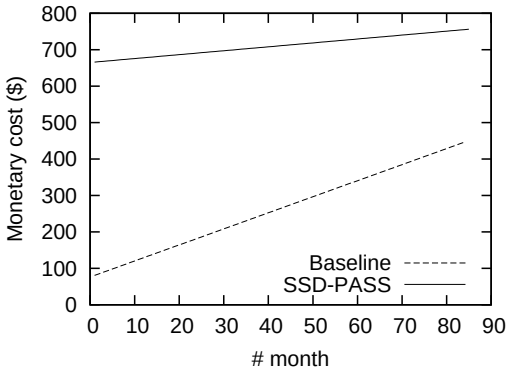


(a) MadBench - 2GB

(b) MadBench - 4 GB



(c) MadBench - 8 GB



(d) MadBench - 16 GB

Figure 5.28: Monetary costs in Isolated Workloads and MadBench.

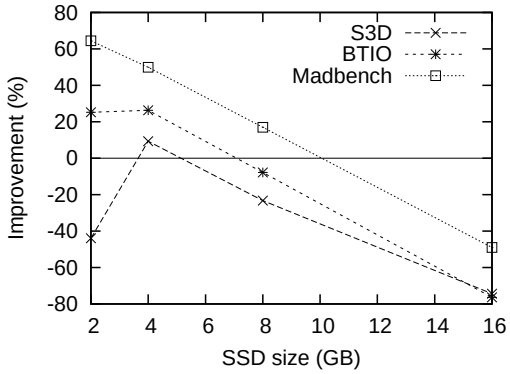


Figure 5.29: Monetary cost improvement in isolated workloads for different SSD devices sizes.

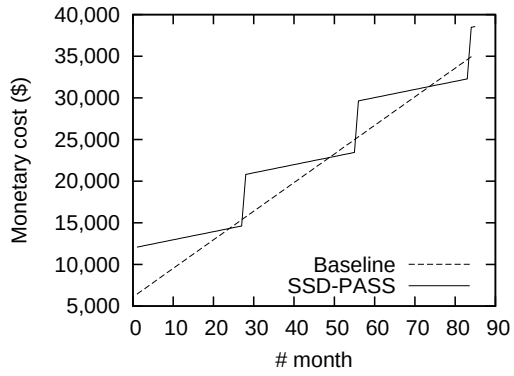
of start/stop disk cycles reached at those times. As S3D writes about 1.5 GB every 15 minutes, SSDs are filled frequently. This causes that disks also wake up in a frequent fashion, reaching the maximum number of duty cycles. BTIO and Madbench write about 1.5 GB every 45 and 96 minutes respectively, prompting the fact that SSDs are not filled so

often, and not reaching the maximum number of duty cycles within the seven year period. None of the traces reach the maximum number of erase operations in the SSDs, not causing their replacement.

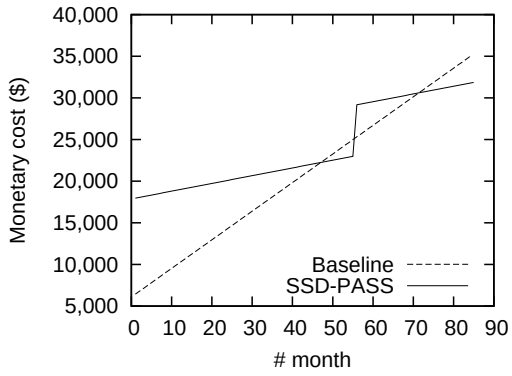
Figure 5.29 shows the monetary cost improvements for our power aware approach relative to the baseline approach. Improvements are shown for the three traces and for SSDs sizes from 2 to 16 GB. As we previously commented, best results are reached with Madbench, as this application writes the less amount of data in less time. The percentages in each trace correspond with monetary costs shown in figures 5.26, 5.27, and 5.28.

5.2.1.2 Working Day Workloads

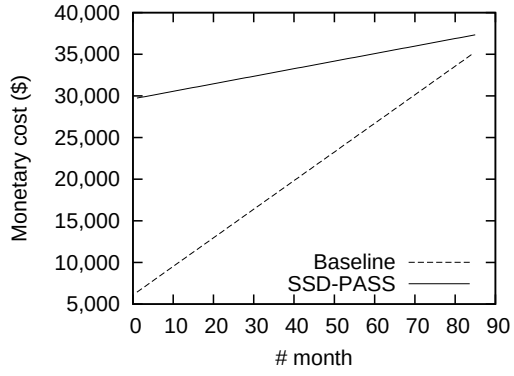
In the second scenario, we execute a mixture of the three traces in the course of one working journey. In this scenario each application use the storage subsystem in a dedicated way. In every combination, each application executes for a certain percentage of time. For example, the S3D 60% - BTIO 20% - Mad 20% combination means that S3D executes 60% of time, BTIO executes 20%, and MadBench the remaining 20% of the time.



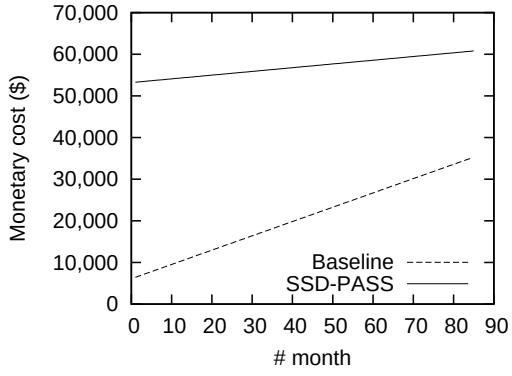
(a) S3D 60% - BTIO 20% - Mad 20% - 2GB



(b) S3D 60% - BTIO 20% - Mad 20% - 4 GB



(c) S3D 60% - BTIO 20% - Mad 20% - 8 GB

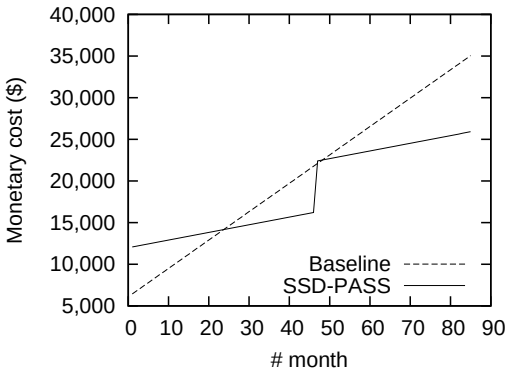


(d) S3D 60% - BTIO 20% - Mad 20% - 16 GB

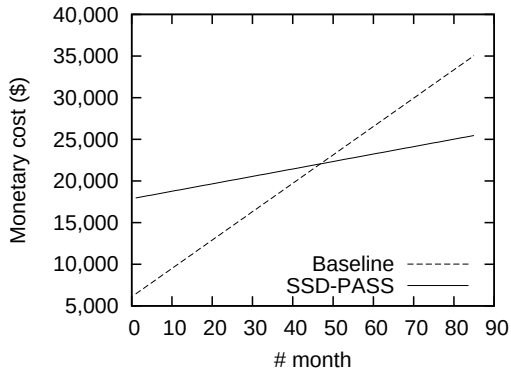
Figure 5.30: Monetary costs in Working Day Workloads and S3D 60% - BTIO 20% - MadBench 20%.

Amortization costs for the S3D 60% - BTIO 20% - Mad 20% combination are shown

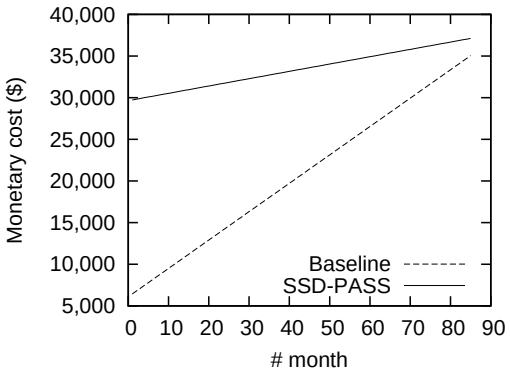
in figure 5.30. Improvements are shown for SSDs sizes from 2 to 16 GB. Using 2 GB sized SSDs, improvements are never reached. That is because, every 27 months new disks are purchased, exceeding the budget of the baseline approach. That happens due to the small size of SSDs, which are filled more frequently, and also make disk drives start/stop more frequently. Using 4 GB sized SSDs, amortization is reached at month 48, but the purchase of new disks at month 56 exceeds the budget of the baseline approach until month 72. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult to amortize. Using 4GB sized SSDs we obtain an energy consumption improvement of 9.06%.



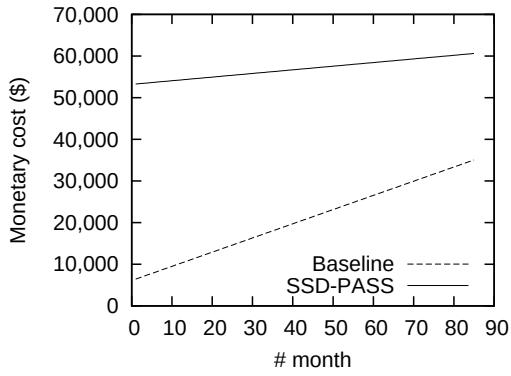
(a) S3D 20% - BTIO 60% - Mad 20% - 2GB



(b) S3D 20% - BTIO 60% - Mad 20% - 4 GB



(c) S3D 20% - BTIO 60% - Mad 20% - 8 GB

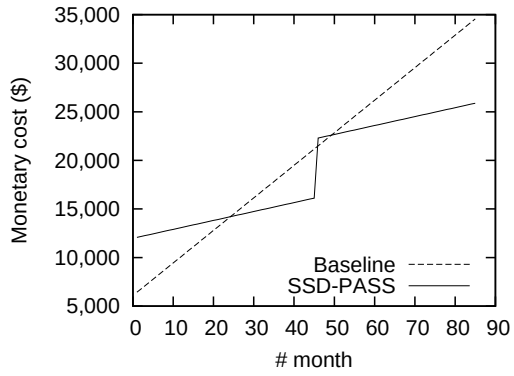


(d) S3D 20% - BTIO 60% - Mad 20% - 16 GB

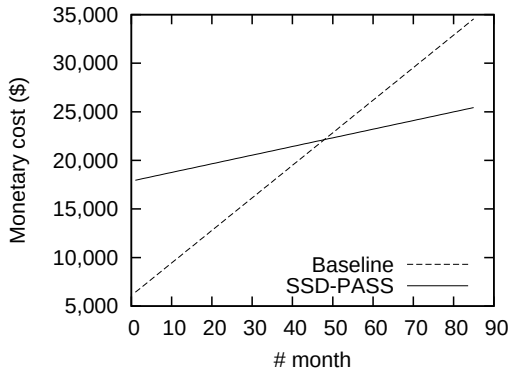
Figure 5.31: Monetary costs in Working Day Workloads and S3D 20% - BTIO 60% - MadBench 20%.

Amortization costs for S3D 20% - BTIO 60% - Mad 20% are shown in figure 5.31. Improvements are shown for SSDs sizes from 2 to 16 GB. Using SSDs of size 2 GB, amortization is reached at month 24. Purchasing new disks at month 47 means exceeding the budget of the baseline approach, until month 49. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 4 GB sized SSDs, amortization is reached at month 47. From that moment on, monetary costs in our

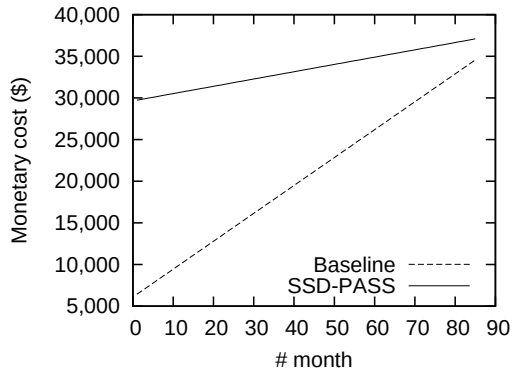
power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult the purchase of SSDs to amortize. Using SSDs of size 4 GB we obtain an energy consumption improvement of 26.91%.



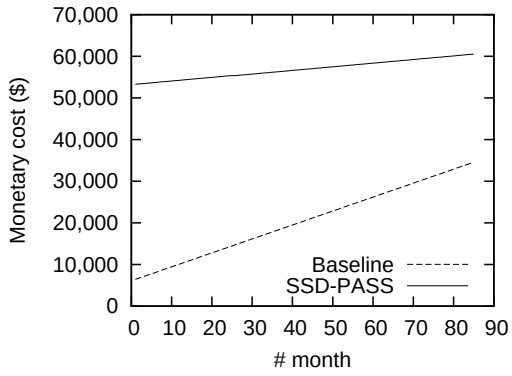
(a) S3D 20% - BTIO 20% - Mad 60% - 2GB



(b) S3D 20% - BTIO 20% - Mad 60% - 4 GB



(c) S3D 20% - BTIO 20% - Mad 60% - 8 GB



(d) S3D 20% - BTIO 20% - Mad 60% - 16 GB

Figure 5.32: Monetary costs in Working Day Workloads and S3D 20% - BTIO 20% - MadBench 60%.

Amortization costs for S3D 20% - BTIO 20% - Mad 60% are shown in figure 5.32. Improvements are shown for SSDs sizes from 2 to 16 GB. Using 2 GB sized SSDs, amortization is reached at month 25, but the purchase of new disks at month 46 exceeds the budget of the baseline approach until month 50. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. Using 4 GB sized SSDs, amortization is reached at month 48. From that moment on, monetary costs in our power aware approach are smaller than in the baseline approach. Using 8 and 16 GB sized SSDs, amortization is never reached. That is because prices per Giga in SSDs are still very high, and the bigger the sizes, the higher the prices, and the more difficult the purchase of SSDs to amortize. Using 2 GB sized SSDs we obtain an energy consumption improvement of 24.65%.

Figure 5.33 shows how the different mixtures of the traces affect monetary cost im-

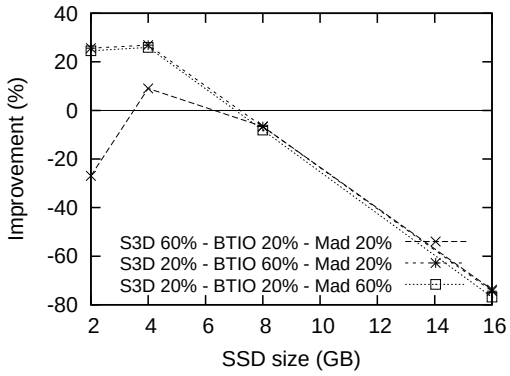


Figure 5.33: Monetary cost improvement in Working Day Workloads for different SSD devices sizes.

provements for SSDs sizes from 2 to 16 GB. The key indicates the time percentage that each trace executes in every combination. Combinations with the highest number of executions of Madbench and BTIO significantly outperform the combination with the highest number of executions of S3D. This is because S3D almost writes the same amount of data than the other two traces in much less time, which causes more spin ups/downs, and disk replacements.

5.2.1.3 Concurrent Workloads

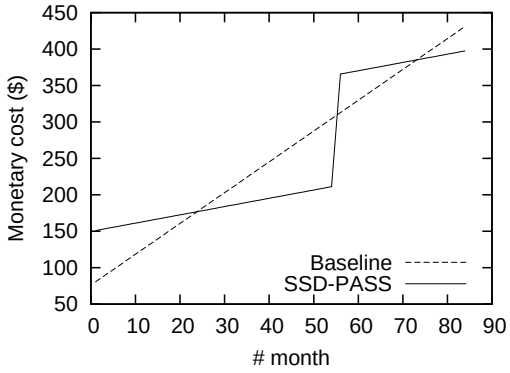


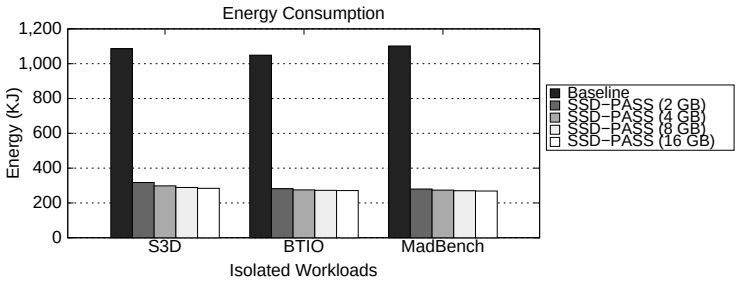
Figure 5.34: Monetary costs in Concurrent Workloads for 4GB sized SSDs.

In the third scenario, we execute BTIO and MadBench concurrently, and the storage system is not accessed in a dedicated way. For every flash device, each application corresponds to a dedicated partition. We do not consider S3D because when is used in a dedicated way, it already fills SSDs very often, causing frequent disk replacements. This suggests that the best option is not putting it altogether with any other applications.

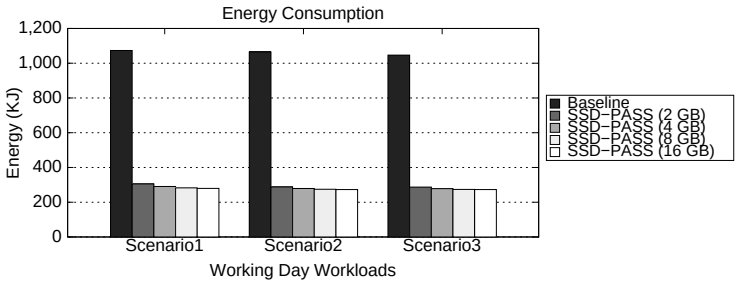
In consideration of the previous results, where 2 and 4 GB sized SSDs performed better than bigger sizes for individual executions of BTIO and Madbench, we used SSDs of size 4 GB. We used equal SSD size partitioning. With 4 GB sized SSDs, the actual size

of the BTIO and Madbench partitions is 2 GB. Individual executions of both traces with 2 GB sized SSDs obtained reasonable monetary cost improvements. This led us to believe that partitioning equally SSDs size 4 GB would give us some leeway in the monetary cost improvements.

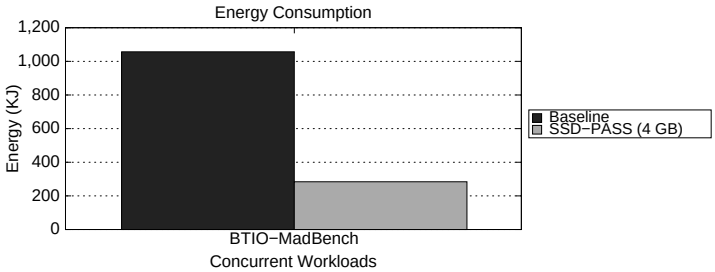
Using the described configuration, we obtain a monetary cost improvement of 8.501%. Amortization costs for the described configuration is shown in figure 5.34. Using 4 GB sized SSDs, amortization is reached at month 24, but the purchase of new disks at month 57 exceeds the budget of the baseline approach until month 74. Starting from that point, monetary costs in our power aware approach are smaller than in the baseline approach. During the execution, we tracked the amount of data written to each partition for every spin-down period. Since not always the same application causes its partition capacity fill and both applications request data in well predictable patterns, we plan to design a dynamic partitioning algorithm which makes it possible to set partition sizes up on run-time.



(a) Isolated workloads



(b) Working day workloads



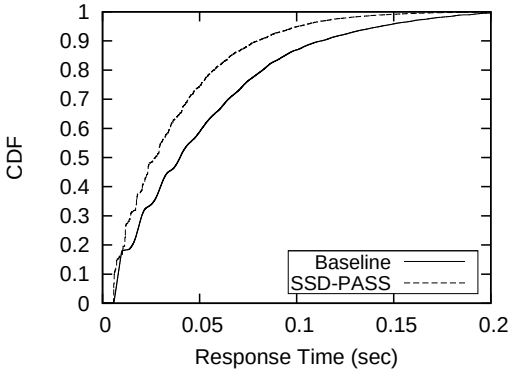
(c) Concurrent workloads

Figure 5.35: Mean energy consumption for a storage element under different executions of isolated, working day, and concurrent workloads.

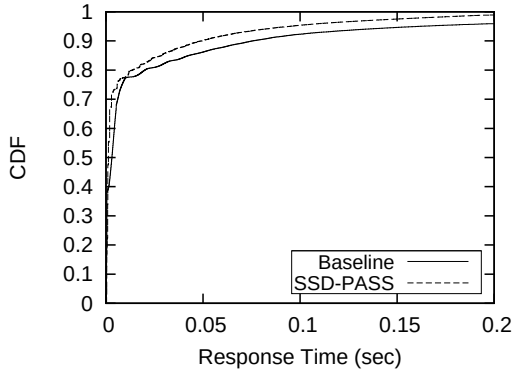
5.2.1.4 Energy consumed

In order to give an idea of the energy savings obtained when SSD-PASS is applied, Figure 5.35 shows the mean energy consumed by a single storage element, under all the previously described scenarios. Energy consumptions are shown for a one day period. As can be seen, for almost all the cases energy consumptions are around 3.5 times lower in SSD-PASS than in the Baseline approach, where no energy optimizations are applied.

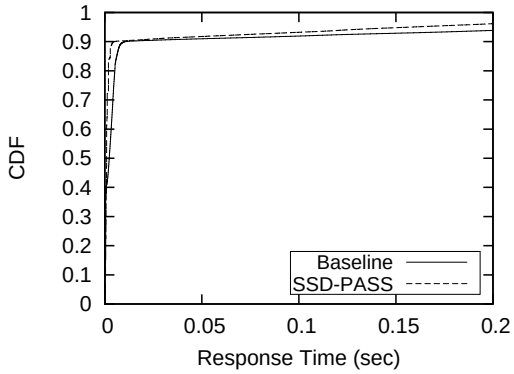
5.2.1.5 Performance Evaluation



(a) Isolated workload - S3D



(b) Working day workload - S3D 60% - BTIO 20% - Mad 20%



(c) Concurrent workload - BTIO and Madbench

Figure 5.36: Performance evaluation for the worst cases of isolated, working day, and concurrent workloads.

We use the cumulative distribution functions (CDFs) of response times as the performance metric in our comparison. Figure 5.36 shows how the worst cases in different scenarios affect SSD-PASS performance. For a certain value of x in every trace, SSD-PASS presents greater percentages in which response times are lower or equal to x. That is because of the hits on the SSDs, which provides better performance gains. We chose to show worst cases to demonstrate that there is little benefit in performance even for those cases.

For the Isolated workload - S3D scenario and the SSD-PASS approach nearly 90% of the response times are shorter than 0.079 secs. For the Baseline approach and the same scenario, approximately 78% of the response times are shorter than 0.079 secs. For the working day workload shown in Figure 5.36(b) and the SSD-PASS approach, 90% of the response times are shorter than 0.049 secs. Unlike the SSD-PASS approach, in the Baseline approach and the same scenario, approximately 86% of the response times are shorter than 0.049 secs. For the concurrent workload scenario, 90% of the response times are shorter than 0.00455 secs. On the Baseline approach and the same case, 74% of the response times are lower than 0.00455 secs.

5.2.2 Prefetching evaluation

In the next two Subsections, 5.2.2.1 and 5.2.2.2, the application of the Sequential read-ahead policy, described in Section 4.4.1, is analyzed. Also, an economic analysis is provided in Subsection 5.2.2.3 for the Offline read-ahead policy, previously described in Section 4.4.4.

5.2.2.1 Synthetic workloads

We have implemented a synthetic benchmark which simulates the behavior of data-intensive applications. The benchmark consists of a configurable number of alternating computation and I/O phases. The compute phases are simulated by idle spinning. In the I/O phase a process reads non-overlapping records from a file. In this experiment, the block size used was 256 KBytes and the storage system counted with 10 magnetic disks, in which file blocks are mapped round-robin over all disks. The benchmark was configured to read a total of 512 MBytes, i.e. in each iteration a process reads a record of 4 MBytes in 128 iterations. The configurable parameters of this benchmark are the file access interval times and the stripping factor.

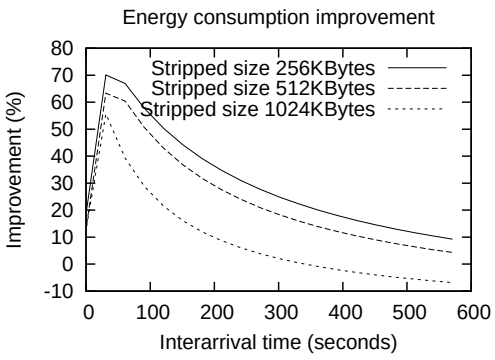


Figure 5.37: Power-saving improvement obtained with different data layouts.

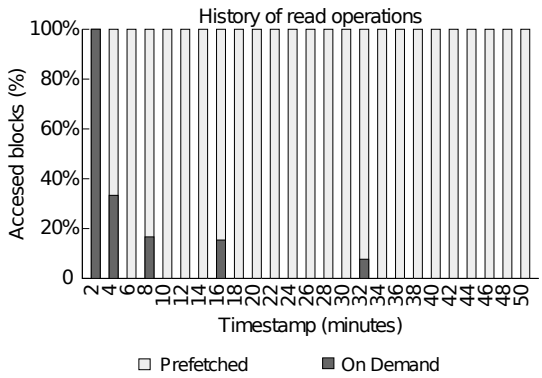


Figure 5.38: Prefetch hit ratio obtained with interarrival times of 120 seconds.

Figure 5.37 compares the benefits of our sequential read-ahead policy with different file system's stripping factors. We observe that our solution outperforms the baseline approach up to inter arrival access times of 300 seconds in case of 1024 KBytes stripping size.

Additionally, the best results are obtained when the block size is equal to the stripping size, as all disks are accessed during each read operation. We demonstrate in Figure 5.38 that our prefetching mechanism increases idle periods on disks, avoiding misses as much as possible. The number of on demand requests decreases due to our prefetching solution increases windows size in each iteration. Finally we conclude from the evaluation that SSD-PASS may bring substantial benefits for bursty workloads.

5.2.2.2 Realistic workloads

To evaluate our storage system, we have used two realistic data-intensive I/O workloads. Financial1 [uma11] is the I/O core of an OLTP application gathered at a huge financial organization. It performs about 5 million requests over 24 disks. Cello99 [Cel11] is a shared compute/mail server from HP Labs. It performs about 6 million requests over 25 disks.

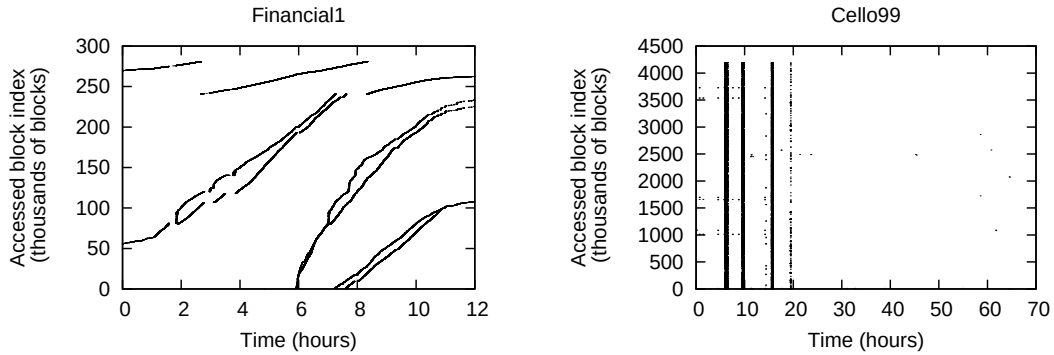


Figure 5.39: Accessed Block index for read operations in Financial1 and Cello99 workloads. Requests in the same timestamp correspond with bursty files accesses.

Figure 5.39 illustrates the regions accessed in the file for Financial1 and Cello99 at disk 1. We initially measured the sequentiality ratio as the percentage of read requests that are sequential in a certain period. The Financial1 presents a sequential ratio of 79% of the total read requests. For Cello99 sequentiality is reached is about 5%.

Figure 5.40 plots read miss accesses at disk 1 of Financial1 workload using the Sequential read-ahead policy. We observe that the density of miss accesses decreases, comparing the accessed block index of Figure 5.39. Financial1 achieves nearly 51% of energy savings as shown in Figure 5.41. Benefits were obtained because of the highly sequential read access patterns at some of the disks and because our prefetching algorithm is able to detect future requests. However, energy consumption for Cello99 is higher than in the Baseline approach. This is due to the low level of sequentiality in Cello99.

Figure 5.42 plots the cumulative distribution function curves (CDF) for response times of both workloads for disk 0. For a certain value of x in every workload, SSD-PASS approach presents greater percentages in which response times are lower or equal to x. That is because of read hits on the SSD, which provides better performance gains. For a small fraction of the requests, we observe that in some cases our solution obtains service times of over 15 seconds. Such requests are the ones which have to wait for a disk to spin-up.

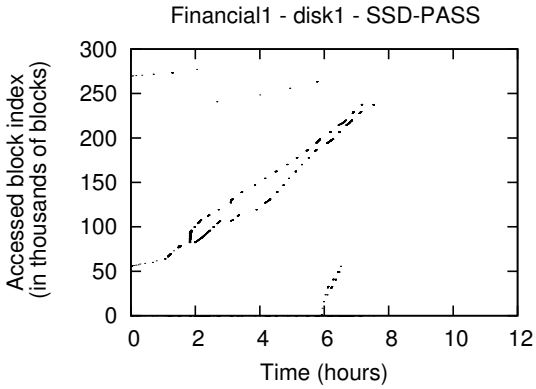


Figure 5.40: Misses at disk 1 for the accessed blocks of the Financial1 workload.

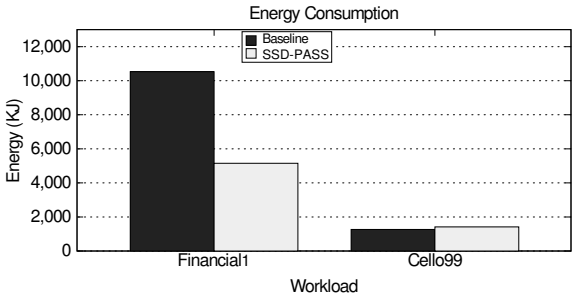


Figure 5.41: Summary of energy consumption obtained for Financial1 and Cello99 worloads.

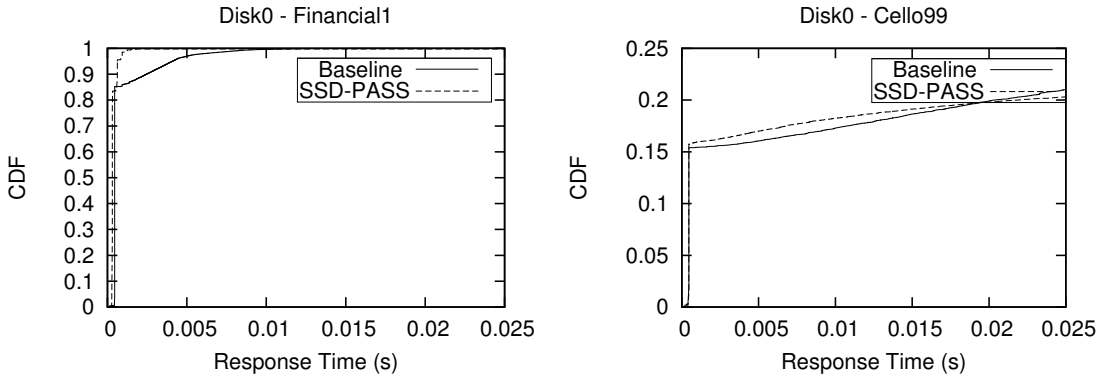


Figure 5.42: CDF curves for response times in Financial1 and Cello99 workloads at disk 0.

5.2.2.3 Economic based Evaluation

As shown in the previous section, the Sequential read-ahead policy works well for most sequential applications. However, as shown in Section 4.4, even for sequential traces, its application and its extensions do not guarantee the fact that disks' spin-ups break long idle intervals. Taking this into account, monetary costs in the long term may be higher in the power aware approaches than in the baseline ones due to constant disk replacements, and the subsequent purchasings of new disks.

The Offline read-ahead policy, whose knowledge is based on previous executions, uses non-blind prefetching and replacement policies, that let disk drives stay in the *standby* state for longer.

Here, we describe an economic based evaluation for the The Offline read-ahead policy.

Amortization costs for Cello99 and Financial are shown in figure 5.43. Using 2 GB sized SSDs, amortization is reached at month 24. Starting from that point, monetary costs in our power aware approach is smaller than in the baseline approach. Unlike the previously

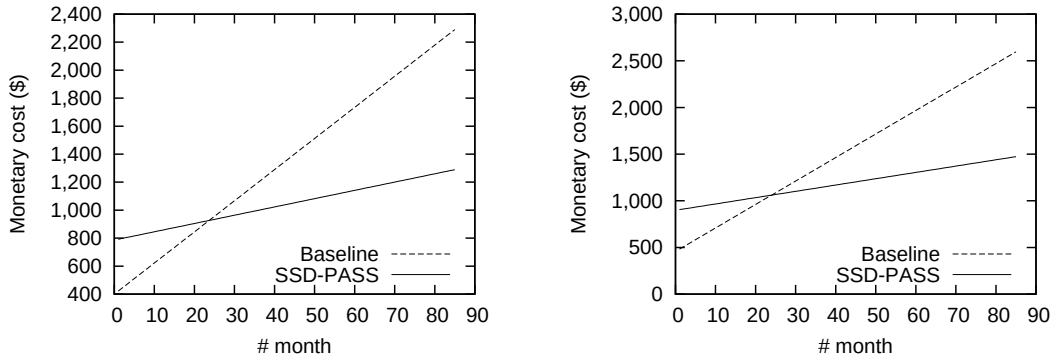


Figure 5.43: Monetary costs in Isolated Workloads and Cello99 (left) and Financial (right).

analyzed traces (S3D, BTIO, and MadBench), Cello99 and Financial do not write so many data, and spin-ups are not so frequent. This makes that purchasing of new disks do not happen, and monetary costs are lower.

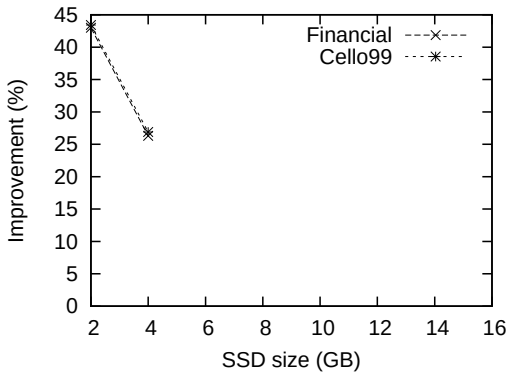
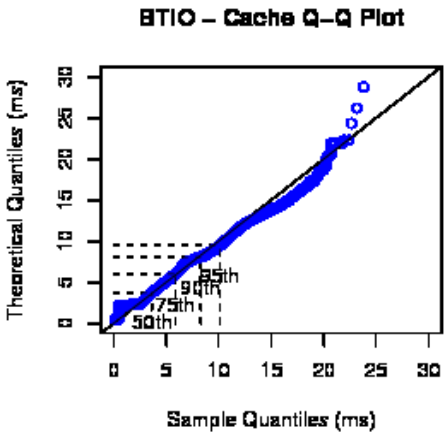


Figure 5.44: Monetary cost improvement in Isolated Workloads for 4 GB sized SSDs.

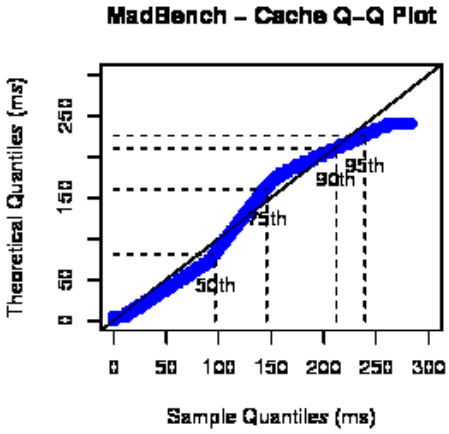
Figure 5.44 shows the monetary cost improvement for our power aware approach relative to the baseline approach. Improvements are shown for the two traces and for SSD devices sizes from 2 to 4 GB. Using SSDs of size 2 GB we obtain energy consumption improvements of 42% and 43% for Financial and Cello99, respectively.

5.3 Using black-box modeling for energy-aware techniques

In this section, the previously proposed black box model approach is used to measure energy consumption in hard disk drives. Models are constructed for several of the previous traces, and for the Seagate Cheetah 15K.5 hard disk, which is the one used in the energy aware evaluations.



	type	π	μ	σ
Distribution 1	LogNormal	-	1.71	0.39
Distribution 2	Constant	-	0.46	-

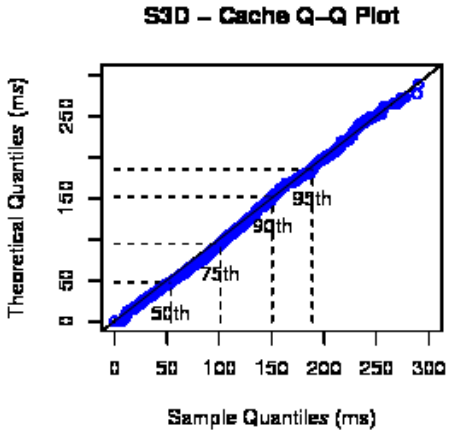


	type	π	μ	σ
Distribution 1	Uniform	-	0	150
Distribution 2	Uniform	-	150	300

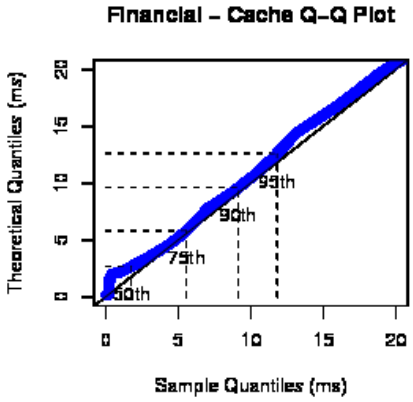
Figure 5.45: Q-Q plots for response times from a Seagate Cheetah 15K.5 disk, and a BTIO and a MadBench workloads. The tables show the parameters of the modeled distributions.

Figure 5.45 shows Q-Q plots for BTIO and MadBench workloads, and tables containing parameters of the modeled distributions. For BTIO, two distributions are identified. Response times from Distribution 1 are generated when writes are not recorded in the disk cache, and reads are serviced from the disk platters. Response times are generated from Distribution 2 when the current request is serviced from the disk cache. For MadBench, two distributions are also identified. Response times from Distribution 1 are generated when the current request arrives after a period of idleness. Response times are generated from Distribution 2 when the current request arrives, and one or several previous requests have not been serviced yet.

Figure 5.46 shows Q-Q plots for S3D and Financial workloads, and tables containing parameters of the modeled distributions. For S3D, just one distribution is identified, and all response times are generated from it. For Financial, two distributions are identified.



	type	π	μ	σ
Distribution 1	Exponential	-	69.26	-



	type	π	μ	σ
Distribution 1	Constant	-	0.196	-
Distribution 2	LogNormal	-	1.62	0.66

Figure 5.46: Q-Q plots for response times from a Seagate Cheetah 15K.5 disk, and a S3D and a Financial workloads. The tables show the parameters of the modeled distributions.

Response times are generated from Distribution 1 when the number of blocks requested in the current request are more than 60, or the current request must wait for previous requests to be serviced. Response times are generated from Distribution 2, when requests are serviced from the disk cache.

	Demerit	Energy DiskSim	Energy BBM	Energy error
BTIO	7.25%	33290.29 J	33306.20 J	0.04%
MadBench	11.63%	68933.28 J	68958.55 J	0.036%
S3D	7.36%	10629.27 J	10680.77 J	0.48%
Financial	15.58%	522704 J	524189.6 J	0.28%

Table 5.7: Demerits, values of energy, and energy errors, in relative terms, for our black box model approach (bbm) and DiskSim.

As it is shown in Table 5.7, demerits are not very high, and that is why, values of

energy are very similar, both in DiskSim and in our black box model (bbm). This lead us to conclude that our black box model can be used for energy aware issues.

5.4 Summary

This chapter have presented an experimental evaluation, both of the previously described black box model for the disk drive, and of the proposed energy aware techniques.

For the black box model, our experimental results show that our service time measurement tool, *play*, is validated in Section 5.1.1. Accuracy in the construction of new models, for several real workloads, is shown in Section 5.1.2. A comparison between our black box model and DiskSim, shows that our black box model can be as accurate as detailed models, and that it brings a substantial performance benefit.

Results for non-trained workloads are shown in Section 5.1.3. The experiments demonstrate the accuracy of our black box model for workloads with similar characteristics of size, queuing times, and sequentiality than one of the previously trained traces, used in the model's construction.

Results for models based on synthetic traces are shown in Section 5.1.4. The experiments demonstrate that models based on synthetic traces may be less accurate, but also more general and versatile. Results are shown for workloads with similar characteristics of size and sequentiality than the trained synthetic trace.

For power aware techniques, our experimental results in Section 5.2.1, show that our write-buffering policy reduces energy consumption, and that those reductions have effects into monetary cost reductions. Different sizes for supportive SSDs have been evaluated, and the experiments demonstrate that architectures that use SSDs bigger than 8 GB do not even amortize initial budgets. This is due to the still high prices of SSDs.

Results for our power aware prefetching policies are shown in Section 5.2.2. Experiments for synthetic and realistic workloads demonstrate that the Sequential read-ahead policy works well for sequential applications. An economic based evaluation for the Offline read-ahead policy shows monetary costs reductions in the long term, even for non-sequential applications.

Results in Section 5.3 show that black box modeling techniques can be used for accurate energy simulations.

Chapter 6

Final remarks and conclusions

In this thesis we have proposed a black box model for disk drives, based on probability distributions, and a generic I/O power saving architecture for HPC applications.

The proposed black box model constructs new disk models by using service times obtained from a real disk. To obtain service times from any kind of disk, we implemented a new tool that uses standard POSIX calls. The sequence of obtained service times is seen as a sequence of random variables in a stochastic process, fitting one or several probabilistic distributions. Thus, the model generates service times, as random values accordingly with that distributions.

The proposed I/O power saving architecture lies on an hybrid architecture, in which each I/O node consists of an SSD, and a conventional magnetic disk drive. It aims to use SSD devices as block caches for the disk drives. It consists of two main modules: Write-buffering and prefetching modules. New algorithms have been proposed for both modules.

We have shown that black box models for disk drives may be as accurate as detailed models, and also faster. We have also shown that using SSDs as supportive devices for disk drives, provides significant energy savings, and that black box models may be used for accurate energy simulations.

The thesis has properly fulfilled all the primary objectives indicated in Section 1.2. Our approaches have accomplished the thesis objectives in the following ways:

- **Scalability**. The proposed black box model is fast in terms of performance. This is demonstrated in Section 5.1.2, where it is shown that the proposed black box model can be until 600 times faster than other accurate detailed models.

- **Accuracy**. The proposed black box model may be as accurate as detailed simulators. This has also been demonstrated in Section 5.1.2. Our models have been compared with the reference simulator, DiskSim, and it is shown that accuracy values keep roughly on the same range. Moreover, we proposed a new service time measurement

tool in Section 3.4, and the results shown in Section 5.1.1, demonstrate that it fits other previous reference tool.

- **Saving power in hybrid storage systems**. In Chapter 4, we proposed a generic I/O architecture that uses SSDs as supportive devices for disk drives. It works at block level, and aims to use SSDs as caches for disk drives. This configuration is transparent to the file system, and the layers above.

- **Disk Reliability**. The proposed power-aware architecture takes into account disk failures. Analyzed monetary costs include prices of purchasing new disks, as a result of replacing the ones that have reached their maximum number of star/stop cycles.

- **Flash Reliability**. The proposed power-aware architecture takes into account SSDs failures, by mostly using SLC typed SSDs. This reduces the likelihood for errors, and also their replacements, and subsequent monetary costs.

- **High-Performance**. Although some tradeoffs exist in power saving solutions, applications get the most of SSD devices, by requesting data to SSDs, as much as possible. This has been demonstrated in Sections 5.2.1.5, and 5.2.2.2

- **Economic Feasibility**. The proposed architectures reflect monetary costs improvements, in different scenarios. This has been shown in Sections 5.2.1, and 5.2.2.3.

6.1 Contributions

This thesis makes the following contributions:

- **Service Time Measurement Tool**. This thesis presents a new service time measurement tool, in order to get a measurement for every request, as a prior task to build a simulation model. It gets measurements by using standard POSIX calls and not SCSI commands. That is why, it performs response time measurements on any kind of disk, and hence models any disk, with any kind of interface. It is described in Section 3.4.

- **Black box Model for Hard Disk Drives**. This thesis presents and evaluates a new black box model, for hard disk drives. It is based on probabilistic distributions. It has been demonstrated that it is faster than another referenced detailed models, and also, can be as accurate as them. Two techniques have been proposed. The first one, based on realistic workloads, is very accurate for applications with similar characteristics of size, queuing times and sequentiality, than the trained applications. The second one is based on synthetic workloads. Queuing times are calculated at simulation stages. This approach is accurate for applications with similar characteristics of size and sequentiality that the trained synthetic workload.

- **SSD-based Power-aware Storage System architecture**. This thesis presents and evaluates a general power saving-aware architecture for large parallel computing

environments such as clusters or supercomputers. It lies on an hybrid architecture, in which each I/O node consists of a SSD and a conventional magnetic disk. Our general procedure aims to use a SSD as a block cache for a specific disk on I/O nodes. I/O operations are transparent for both SSD devices, and file systems, and other layers above.

- **Write-buffering policy**. This thesis presents and evaluates a new write-buffering policy, that is integrated in the previously described architecture. It redirects writes to SSDs, as much as possible, as long as SSDs have enough space. It also gets the most of the repetitive patterns from HPC applications, by avoiding disk accesses, that come from read requests that can be serviced from SSDs. An analysis of monetary costs is provided for this approach in Section 5.2.1.

- **Prefetching policies**. This thesis presents four new prefetching policies, that can be integrated in the previously described architecture. The first three approaches are dynamic, and work specially well for sequential applications. As they are blind, and do not count on previous information, they make excessive spin-ups and, hence, disk replacements. As a result of this, a forth prefetching policy was proposed. It is offline, and counts on information from previous executions. It is based on a band detection method, to detect specific accessed areas from disk drives, and move them to SSDs. This last approach works well both for sequential and non sequential applications. An analysis of monetary cost is also provided for this approach in Section 5.2.2.3.

6.2 Thesis results

The principal contributions of the thesis have been published in diverse papers in international conferences and journals. We enumerate the publications classified in three groups: articles in journals, international and national conferences.

- Journals
  - *Power Saving aware prefetching for SSD-based systems.* Laura Prada, Javier Garcia, J. Daniel Garcia, Jesus Carretero. The Journal of Supercomputing. Springer. March, 2011. Impact Factor: 0.687.

- International Conferences
  - *A Black Box Model for Storage Devices based on Probability Distributions.* Laura Prada, Alejandro Calderón, Javier García, J. Daniel Garcia, Jesus Carretero. 10th IEEE International Symposium on Parallel and Distributed Processing with Applications. July, 2012.
  - *A Power-aware Based Storage Architecture for High Performance Computing.* Laura Prada, Javier Garcia, J. Daniel Garcia, Jesus Carretero, Alberto Nuñez. 13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011). September, 2011.

  - *Using Write Buffering and Read Prefetching Between Flash and Disk Drives to Save Energy in an Hybrid System.* Laura Prada, J. Daniel Garcia, Jesus Carretero. 16th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010). July, 2010.
  - *Power Saving-aware Solution for SSD-based Systems.* Laura Prada, J. Daniel Garcia, Jesus Carretero, Javier Garcia Blas. International Conference on Mathematical Methods in Science and Engineering (CMMSE 2010). Almeria, Spain. June, 2010.
  - *Saving power in flash and disk hybrid storage system.* Laura Prada, Jose Daniel Garcia, Jesus Carretero, and Felix Garcia. 17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'09). London, England. September, 2009.
  - *Using black-box modeling techniques for modern disk drives service time simulation.* Jose Daniel Garcia, Laura Prada, Javier Fernandez, Jesus Carretero, Alberto Nunez. The 41th Annual Simulation Symposium (ANSS'08). April, 2008.

- National Conferences.
  - *Ahorro energetico en un sistema de almacenamiento hibrido compuesto por un disco duro y varias memorias flash.* Laura Prada, J. Daniel Garcia, Jesus Carretero, and Felix Garcia. Actas de las XX Jornadas de Paralelismo. La Coruña, Spain. September, 2009.
  - *Modelado estocastico de las operaciones de entrada/salida sobre un disco.* Laura Prada, J. Daniel Garcia, Alberto Nuñez, Javier Fernandez, Jesus Carretero, Ramon J. Flores. II Congreso Espaol de Informtica (CEDI 2007). XVIII Jornadas de Paralelismo. Zaragoza, Espaa. September, 2007.

Other achievements of this thesis include research stays and research grants:

- Research stays
  - Departament of Electrical and Computer Engineering at Texas A&M University. Hosted by A.L. Narasimha Reddy. Fall 2008. College Station (USA). Duration: 3 months.
  - Departament of Electrical and Computer Engineering at Texas A&M University. Hosted by A.L. Narasimha Reddy. Summer 2010. College Station (USA). Duration: 3 months.

- Research grants
  - Grant for PhD students mobility, November 2008. Grant funded with 2,135 euros by University Carlos III of Madrid for a 91 days internship at Texas A&M University.
  - Grant for PhD students mobility, May 2010. Grant funded with 3,115 euros by University Carlos III of Madrid for a 89 days internship at Texas A&M University.

6.3 Future directions

There are several lines of research arising from this work that could be pursued:

We plan on designing another algorithms of prefetching/write-buffering for power saving architectures. Specially the ones that combine arrays of disks with one or several SSDs.

We also plan on applying our black box method for SSDs. As was described, SSDs present different characteristics from disk drives. One remarkable characteristic is the asymmetrical performance that some devices present, depending on the type of the request (read/write).

Also, we plan on applying our black box method for arrays of disks/SSDs, and another environments such as SANs and LVMs.

Bibliography

[ABJ+09] Deepak Ajwani, Andreas Beckmann, Riko Jacob, Ulrich Meyer, and Gabriel Moruz. On computational models for flash memory devices. In *Proceedings of the 8th International Symposium on Experimental Algorithms*, pages 16–27, Berlin, Heidelberg, 2009. Springer-Verlag.

[And01] Eric Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-4, HP Laboratories, 2001.

[apa11] Apache HTTP Server Projet, 2011. Available at http://httpd.apache.org.

[APW+08] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[BAT06] Avraham Ben-Aroya and Sivan Toledo. Competitive analysis of flash-memory algorithms. In Yossi Azar and Thomas Erlebach, editors, *Algorithms ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 100–111. Springer Berlin / Heidelberg, 2006.

[BBL06] T. Bisson, S.A. Brandt, and D.D.E. Long. Nvcache: Increasing the effectiveness of disk spin-down algorithms with caching. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*, pages 422 – 432, sept. 2006.

[BBL07] Timothy Bisson, Scott A. Brandt, and Darrell D. E. Long. A hybrid disk-aware spin-down algorithm with i/o subsystem support. In *IPCCC*, pages 236–245, 2007.

[BCFP+06] Gregory Basheda, Marc Chukpa, Peter Fox-Penner, Johannes Pfeifenberger, and Adam Schumacher. Why are electricity prices increasing?, 2006. Available at http://www.edisonfoundation.net/Brattle_report_Web.pdf.

[BCS+08] Surendra Byna, Yong Chen, Xian-He Sun, Rajeev Thakur, and William Gropp. Parallel I/O prefetching using MPI file caching and I/O signatures. In *SC '08*, pages 1–12, 2008.

[BD11] S. Boboila and P. Desnoyers. Performance models of flash-based solid-state drives for real workloads. In *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pages 1 –6, may 2011.

[Bel66] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5:78–101, June 1966.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[BIC+09] Javier García Blas, Florin Isaila, Jesús Carretero, Robert Latham, and Robert B. Ross. Multiple-Level MPI File Write-Back and Prefetching for Blue Gene Systems. In *PVM/MPI*, pages 164–173, 2009.

[BIC10] Javier Garcia Blas, Florin Isaila, and Jesus Carrtero. *Multi-tier cached I/O architecture for supercomputers*. Lambert Academic Publishing, December 2010.

[BITW07] Andrew Birrell, Michael Isard, Chuck Thacker, and Ted Wobber. A design for high-performance flash disks. *SIGOPS Oper. Syst. Rev.*, 41(2):88–93, 2007.

[BS02] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 55–65, 2002.

[bti11] NAS BTIO Benchmark, 2011. Available at http://www.nas.nasa.gov/Resources/Software/npb.html.

[CCJZ06] Computers Feng Chen, Feng Chen, Song Jiang, and Xiaodong Zhang. Smartsaver: Turning flash drive into a disk energy saver for mobile. In *ISLPED 06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 412–417. ACM Press, 2006.

[CCL+02] Avery Ching, Alok Choudhary, Wei Keng Liao, Rob Ross, and William Gropp. Noncontiguous I/O through PVFS. In *CLUSTER '02*, page 405, Washington, DC, USA, 2002. IEEE Computer Society.

[CD73] E.G. Coffman and P.J. Denning. *Operating Systems Theory*. 1973.

[Cel11] HP Labs Tools and Traces, 2011. Available at http://www.hpl.hp.com/research/ssp/software/.

[CG02] Dennis Colarelli and Dirk Grunwald. Massive arrays of idle disks for storage archives. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[CG09] Jesus Carretero and Jose Daniel Garcia. Scalability in data management. *J. Supercomput.*, 47(3):253–254, 2009.

[Cou06] Storage Performance Council. Spc benchmark-1 (spc-1). official specification, 2006.

[CPB03] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *In Proceedings of the 17th International Conference on Supercomputing*, pages 86–97, 2003.

[CZ08] Feng Chen and Xiaodong Zhang. Caching for bursts (c-burst): let hard disks sleep well and work energetically. In *ISLPED '08: Proceeding of the thirteenth international symposium on Low power electronics and design*, pages 141–146, New York, NY, USA, 2008. ACM.

[Den68] P.J. Denning. The working set model for program behavior. In *Communications of the ACM*, volume 11, 1968.

[DF05] S. Daniel and R. E. Faith. A portable, open-source implementation of the spc-1 workload. In *Proceedings of the The IEEE International Workload Characterization*, 2005.

[dis08] The DiskSim Simulation Environment (V4.0), 2008. Available at http://www.pdl.cmu.edu/DiskSim, last visited on January 18, 2012.

[DIX12] DIXTRAC: Automated Disk Drive Characterization, 2012. Available at http://www.pdl.cmu.edu/Dixtrac/index.shtml.

[DKB95] Fred Douglis, P. Krishnan, and Brian Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *Computing Systems*, pages 121–137, 1995.

[ele11a] Us energy information administration, 2011. Available at http://www.eia.doe.gov/cneaf/electricity.

[ele11b] Europe's energy portal, 2011. Available at http://www.energy.eu/.

[GAN93] K. S. Grimsrud, J. K. Archibald, and B. E. Nelson. Multiple prefetch adaptive disk caching. *IEEE Trans. on Knowl. and Data Eng.*, 5:88–103, February 1993.

[GB07] Binny S. Gill and Luis Angel D. Bathen. Amp: adaptive multi-stream prefetching in a shared cache. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 26–26, Berkeley, CA, USA, 2007. USENIX Association.

[GM05] Binny S. Gill and Dharmendra S. Modha. Sarc: sequential prefetching in adaptive replacement cache. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 33–33, Berkeley, CA, USA, 2005. USENIX Association.

[GPW10] Kevin M. Greenan, James S. Plank, and Jay J. Wylie. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, HotStorage'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association.

[gre11] The Green 500, 2011. Available at http://www.green500.org.

[GSKF03] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut T. Kandemir, and Hubertus Franke. Drpm: Dynamic speed control for power mangagement in server class disks. In *ISCA*, pages 169–179, 2003.

[GWW+99] Gregory R. Ganger, Bruce L. Worthington, Bruce L. Worthington, Yale N. Patt, and Yale N. Patt. The disksim simulation environment version 2.0 reference manual. Technical report, 1999.

[HEH+09] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 10:1–10:9, New York, NY, USA, 2009. ACM.

[HHS05] Hai Huang, Wanda Hung, and Kang G. Shin. Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption. *SIGOPS Oper. Syst. Rev.*, 39(5):263–276, 2005.

[HLS96] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. pages 130–142, 1996.

[HLST11] H.H. Huang, Shan Li, A. Szalay, and A. Terzis. Performance modeling and analysis of flash-based storage devices. In *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pages 1 –11, may 2011.

[hs95] http://www.unix systems.org/. *The Portable Operating System Interface*, 1995.

[IGHZ96] William V. Courtright Ii, Garth Gibson, Mark Holland, and Jim Zelenka. A structured approach to redundant disk array implementation. In *Proceedings of the 2nd International Computer Performance and Dependability Symposium (IPDS '96)*, pages 11–, Washington, DC, USA, 1996. IEEE Computer Society.

[Int98] Understandig the flash translation layer (ftl) specificacion. 1998.

[JA01] Eric Anderson July and Eric Anderson. Simple table-based modeling of storage devices, 2001.

[JS08] Nikolai Joukov and Josef Sipek. Greenfs: making enterprise computers greener by protecting them better. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 69–80, New York, NY, USA, 2008. ACM.

[JW92] David M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, Hewlett-Packard Laboratories, Concurrent Systems Project, March 1992.

[KLW94] R. Karedla, J.S. Love, and B.G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38 –46, mar 1994.

[KRM08] Taeho Kgil, David Roberts, and Trevor Mudge. Improving NAND Flash Based Disk Caches. *SIGARCH Comput. Archit. News*, 36(3):327–338, 2008.

[KT91] Michelle Y. Kim and Asser N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Trans. Comput.*, 40:801–810, July 1991.

[KTGU09] Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *Proceedings of the 2009 First International Conference on Advances in System Simulation*, pages 125–131, Washington, DC, USA, 2009. IEEE Computer Society.

[LBP+09] Jongmin Lee, Eujoon Byun, Hanmook Park, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Cps-sim: configurable and accurate clock precision solid state drive simulator. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 318–325, New York, NY, USA, 2009. ACM.

[LCZ05] Zhenmin Li, Zhifeng Chen, and Yuanyuan Zhou. Mining block correlations to improve storage performance. *Trans. Storage*, 1(2):213–245, 2005.

[LH10] Shan Li and H. Howie Huang. Black-box performance modeling for solid-state drives. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 391–393, Washington, DC, USA, 2010. IEEE Computer Society.

[LKHA94] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *In Proceedings of the 1994 Winter USENIX Conference*, pages 279–291, 1994.

[Loh02] Wei-Yin Loh. Regression trees with unbiased variable selection and interaction detection, 2002.

[LPC+07] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6, July 2007.

[lus11] Lustre File System, 2011. Available at `http://lustre.org`.

[LW04] Dong Li and Jun Wang. Eeraid: energy efficient redundant and inexpensive disk array. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 29, New York, NY, USA, 2004. ACM.

[LZY+11] Zhuo Liu, Jian Zhou, Weikuan Yu, Fei Wu, Xiao Qin, and Changsheng Xie. Mind: A black-box energy consumption model for disk arrays. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –6, july 2011.

[M-S98] Flash-memory translation layer for nand flash (nftl). 1998.

[mad11] Madbench Benchmark, 2011. Available at http://crd.lbl.gov/~borrill/MADbench2.

[Man92] Milton E. Mangal. *Constrained estimation of mixture normal distributions by the EM algorithm*. McMaster Univerisy Press, 1992.

[McN00] Bruce McNutt. *The fractal structure of data reference: applications to the memory hierarchy*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[Mes95] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 1995.

[MM03] Nimrod Megiddo and Dharmendra S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130, Berkeley, CA, USA, 2003. USENIX Association.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998.

[moo11] Moodle, 2011. Available at http://moodle.org.

[MWS+07] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Alice X. Zheng, and Gregory R. Ganger. Modeling the relative fitness of storage. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, pages 37–48, New York, NY, USA, 2007. ACM.

[NDR08] Dushyanth Narayanan, Austin Donnelly, and Antony I. T. Rowstron. Write off-loading: Practical power management for enterprise storage. In *FAST*, pages 253–267, 2008.

[NF04] Edmund B. Nightingale and Jason Flinn. Energy-efficiency and storage flexibility in the blue file system. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[NFG+09] Alberto Nunez, Javier Fernandez, Jose Daniel Garcia, Felix Garcia, and Jesus Carretero. New techniques for simulating high performance mpi applications on large storage networks. *J. Supercomput.*, 2009.

[NTD+09] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 145–158, New York, NY, USA, 2009. ACM.

[omn12] OMNeT++ Community Site, 2012. Online at http://www.omnetpp.org, last visited on January 18, 2012.

[PBCH01] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power*, 2001.

[PJ90] Shane P. Pederson and Mark E. Johnson. Estimating model discrepancy. *Technometrics*, 32:305–314, July 1990.

[PS04] Athanasios E. Papathanasiou and Michael L. Scott. Energy efficient prefetching and caching. In *In Proc. of the USENIX 2004 Annual Technical Conference*, pages 255–268, 2004.

[PWB07] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[r-c11] The R Project for Statistical Computing, 2011. Available at http://www.r-project.org/, last visited on July 26, 2011.

[RAG+03] Xu R., Wang A., Kuenning G., Reiher P., and Popek G. Conquest: Combining battery-backed ram and threshold-based storage scheme to conserve power. *19th Symposium on Operating Systems Principles (SOSP)*, 2003.

[RW94] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *Computer*, 27:17–28, March 1994.

[s3d11] Sandia National Laboratories I/O Traces, 2011. Available at http://www.cs.sandia.gov/Scalable_IO/SNL_Trace_Data.

[Sam03] Samsung Electronics Co. 512M x 8Bit / 256M x 16Bit NAND Flash Memory (K9K4GXXX0M) Data Sheets, 2003.

[Sam05] Samsung Electronics Co. 1G x 8Bit / 2G x 16Bit NAND Flash Memory (K9L8G08U0M) Data Sheets, 2005.

[San04] B. Sanwel. Kernel korner: extending battery life with laptop mode. *Linux Journal*, 2004.

[SG00] Jiri Schindler and Gregory R. Ganger. Automated disk drive characterization. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 112–113, 2000.

[SGLG02] Jiri Schindler, John Linwood Griffin, Christopher R. Lumb, and Gregory R. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 259–274, 2002.

[SGS08] Sriram Sankar, Sudhanva Gurumurthi, and Mircea R. Stan. Sensitivity based power management of enterprise storage systems. In *MASCOSTS*, pages 253–267, 2008.

[Sik07] Jakub Sikora. Content disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?, 2007.

[SMW98] Elizabeth Shriver, Arif Merchant, and John Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint International Conference on Measurement and Modeling of Computer Systems*, pages 182–191, 1998.

[sni11] SNIA Trace Repository, 2011. Available at http://iotta.snia.org/traces.

[SPBW10] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.

[spc11] Storage performance council, 2011. Available at http://www.storageperformance.org.

[Sym99] Symmetrix 3000 and 5000 Enterprise Storage Systems product description guide, 1999. Available at http://www.emc.com/.

[TCG02] P. Triantafillou, S. Christodoulakis, and C. A. Georgiadis. A comprehensive analytical performance model for disk devices under random workloads. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):140–155, 2002.

[TLY06] Hung-Wei Tseng, Han-Lin Li, and Chia-Lin Yang. An energy-efficient virtual memory system with flash memory as the secondary storage. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 418–423, New York, NY, USA, 2006. ACM.

[UAM01] M. Uysal, G.A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 183 –192, 2001.

[UGB+08] Luis Useche, Jorge Guerra, Medha Bhadkamkar, Mauricio Alarcon, and Raju Rangaswami. Exces: External caching in energy saving storage systems. *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, Feb. 2008.

[uma11] UMass Trace Repository, 2011. Available at http://traces.cs.umass.edu.

[VMQ03] Elizabeth Varki, Arif Merchant, and Xiaozhou Qiu. An analytical performance model of disk arrays under synchronous i/o workloads. Technical report, Univ. of New Hampshire, 2003.

[WAA+04] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with cart models. In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 588–595, Washington, DC, USA, 2004. IEEE Computer Society.

[WGP94] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, SIGMETRICS '94, pages 241–251, New York, NY, USA, 1994. ACM.

[WGPW95] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. On-line extraction of scsi disk drive parameters. In *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 146–156, 1995.

[Wil95] John Wilkes. The pantheon storage-system simulator, 1995.

[WOQ+07] Charles Weddle, Mathew Oldham, Jin Qian, An-I Andy Wang, Peter L. Reiher, and Geoffrey H. Kuenning. Paraid: A gear-shifting power-aware raid. *TOS*, 3(3), 2007.

[WR10] Xiaojian Wu and A. L. Narasimha Reddy. Exploiting concurrency to improve latency and throughput in a hybrid storage system. In *IEEE MASCOTS Conf*, 2010.

[WYZ08] Jun Wang, Xiaoyu Yao, and Huijun Zhu. Exploiting in-memory and on-disk redundancy to conserve energy in storage systems. *IEEE Trans. Comput.*, 57(6):733–747, 2008.

[YM95] Philip S. Yu and Arif Merchant. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Trans. Comput.*, 44:419–433, March 1995.

[YNS+08] Jin Hyuk Yoon, Eyee Hyun Nam, Yoon Jae Seong, Hongseok Kim, Bryan Kim, Sang Lyul Min, and Yookun Cho. Chameleon: A high performance flash/fram hybrid solid state disk architecture. *IEEE Comput. Archit. Lett.*, 7:17–20, January 2008.

[YUK06] Li Yin, S. Uttamchandani, and R. Katz. An empirical exploration of black-box performance models for storage systems. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*, pages 433 – 440, sept. 2006.

[ZCT+05] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. In *SOSP*, pages 177–190, 2005.

[ZGQ08] Qi Zhu, Erol Gelenbe, and Ying Qiao. Adaptive prefetching algorithm in disk controllers. *Perform. Eval.*, 65(5):382–395, 2008.

[ZPL01] Yuanyuan Zhou, James F. Philbin, and Kai Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 91–104, 2001.

[ZSG+03] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[ZSZ04] Qingbo Zhu, Asim Shankar, and Yuanyuan Zhou. Pb-lru: A self-tuning power aware storage cache replacement algorithm for conserving disk energy. In *In Proceedings of the 18th International Conference on Supercomputing*, pages 79–88. ACM Press, 2004.