Performance Specific I/O Scheduling Framework for Cloud Storage

A Thesis
Submitted for the Degree of
Master of Science
in the **Faculty of Engineering**

by

NITISHA JAIN

SUPERCOMPUTER EDUCATION AND RESEARCH CENTRE
Indian Institute of Science
Bangalore – 560 012 (INDIA)

MARCH 2015

DEDICATED TO

My Mom, Dad and Sister..

whose love and support is always with me..

Acknowledgements

I am deeply thankful to all those who have contributed towards this work. First of all, I would like to express my sincerest regards to my research guide Dr J. Lakshmi for her continuous support and encouragement throughout the course of my degree. She has been the guiding light for me and without her keen insight, this work would not have taken the shape that it has. The flexibility and freedom to explore and implement my ideas that she provided me is hard to find with many supervisors and I am truly grateful to her. I would also like to express my gratitude towards Prof. Nandy for enabling my research work under his research group by providing the necessary means and resources for the implementation of my proposed ideas. All the members of CADLab also deserve mention for including me in all the fun parties and gatherings and also being available to help out whenever I needed assistance in my work. I am thankful to my seniors Ankit and Mohit for being extremely patient and understanding during the initial phase of my research and mentoring me throughout the time we spent together in Cloud Lab. I would also like to mention all my lab members including Aakriti, Geetha, Pavan, Pratima as well the interns Monisha and Anusha for providing a wonderful and friendly environment in the lab and making the work enjoyable.

Special thanks goes to Prakash whose contribution in my research has been next to only my guide. He has been a wonderful companion during my stay at IISc, helping me out in all aspects of my work from technical to presentation ones. I owe the genesis of my research problem statement to the regular discussions over tea with Prakash and his lab mate Anirudh. These interactions kept me motivated and provided the much needed moral support especially during the difficult initial phase. I am deeply indebted to him for the numerous favours he has happily done for me.

I am also thankful to my many friends in the institute for making my stay here a memorable and cherished one. Thanks to my dear friend Aashish for all the love and support and for having immense faith in my abilities even when I myself was doubtful. I feel blessed to have him as my friend who I can always depend upon and I owe many special memories to him which I dearly cherish. Ashutosh has also been a constant source of inspiration as well as the craziest fun moments in IISc. Also thanks to my friends Saranya and Aadirupa for being always willing to assist me in my work.

Last and most importantly, I cannot express enough gratitude to my family, my loving parents and my little sister Ruchika for being extremely supportive of my work. My dearest mother has been my inspiration and a pillar of strength throughout the course of my degree. Our long daily discussions about work and life have kept me going and enabled me to stay away from home all this time and concentrate on my work. I dedicate this thesis to my family.

"If you wish to make an apple pie from scratch, you must first invent the universe."

– Carl Sagan, Cosmos

Abstract

Virtualization is one of the important enabling technologies for Cloud Computing which facilitates sharing of resources among the virtual machines. However, it incurs performance overheads due to contention of physical devices such as disk and network bandwidth. Various I/O applications having different latency requirements may be executing concurrently on different virtual machines provisioned on a single server in Cloud data-centers. It is pertinent that the performance SLAs of such applications are satisfied through intelligent scheduling and allocation of disk resources.

The underlying disk scheduler at the server is unable to distinguish between the application requests being oblivious to the characteristics of these applications. Therefore, all the applications are provided best effort services by default. This may lead to performance degradation for the latency sensitive applications. In this work, we propose a novel disk scheduling framework *PriDyn* (*Dyn*amic *Pri*ority) which provides differentiated services to various I/O applications co-located on a single host based on their latency attributes and desired performance. The framework employs a scheduling algorithm which dynamically computes latency estimates for all concurrent I/O applications for a given system state. Based on these, an appropriate priority assignment for the applications is determined which is taken into consideration by the underlying disk scheduler at the host while scheduling the I/O applications on the physical disk. The proposed scheduling framework is able to successfully satisfy QoS requirements for the concurrent I/O applications within system constraints. This has been verified through extensive experimental analysis.

In order to realize the benefits of differentiated services provided by the *PriDyn* scheduler, proper combination of I/O applications must be ensured for the servers through intelligent meta-scheduling techniques at the Cloud data-center level. For achieving this, in the second part of this work, we extended the *PriDyn* framework to design a proactive admission control and scheduling framework *PCOS* (*P*rescient *C*loud I/*O* *S*cheduler). It aims to maximize the utilization of disk resources without adversely affecting the performance of the applications scheduled on the systems. By anticipating the performance of the systems running multiple I/O applications, *PCOS* prevents the scheduling of undesirable workloads on them in order to maintain the necessary balance between resource consolidation and application performance guarantees. The *PCOS* framework includes the *PriDyn* scheduler as an important component and utilizes the dynamic disk resource allocation capabilities of *PriDyn* for meeting its goals. Experimental validations performed on real world I/O traces demonstrate that the proposed framework achieves appreciable enhancements in I/O performance through selection of optimal I/O workload combinations, indicating that this approach is a promising step towards enabling QoS guarantees for Cloud data-centers.

"If we knew what it was we were doing, it would not be called research, would it?"
– Albert Einstein

Publications

1. Nitisha Jain, J. Lakshmi, "PriDyn : Enabling Differentiated I/O Services in Cloud using Dynamic Priorities", *IEEE Transactions on Services Computing (Special Issue on Cloud Computing)*, vol. PP, no. 99, 2014.

2. Nitisha Jain, J. Lakshmi, "PriDyn : Framework for Performance Specific QoS in Cloud Storage", *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*, June 27 - July 2, 2014, Alaska, USA.

3. Nitisha Jain, J. Lakshmi, "PCOS : Prescient Cloud I/O Scheduler for Workload Consolidation and Performance", under review for the *13th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA-15)*.

Contents

List of Figures

List of Tables

Chapter 1

Performance on Cloud Storage

Cloud computing has emerged as one of the most popular computing paradigms facilitating on-demand delivery of compute resources on a pay per use basis. Virtualization is one of the chief enabling technologies for the Cloud which provides the means for scalable delivery of services through sharing of physical resources across different users and applications. Cloud computing also assists in the consolidation of applications having varied requirements such as I/O intensive and computation intensive on a single server. However, several multi-tenancy issues have been aggravated by such consolidation use-cases. Various applications executing in shared Cloud environments interfere with each other for resource access, resulting in degradation of performance [1]. This interference emanates from the lack of proper isolation and controls for shared usage of the physical resources such as disk, CPU, memory controllers etc. Such sharing issues may affect the Quality of Service(QoS) requirements of the applications in terms of the performance expectations and lead to violation of Service Level Agreements(SLAs).

The Cloud computing model benefits workloads that have some I/O access since during I/O wait by the applications, the idle CPU cycles can be exploited by other applications. It is essentially this property that motivates the server consolidation case. However, evolution of current virtualization technologies has occurred on systems not designed for concurrent access demanded by the multi-tenancy aspects. This is particularly true in case of I/O devices like the network interfaces and disk controllers and disk devices. I/O devices have traditionally been designed for access through protected modes using OS system calls and most virtualization technologies to date preserve this model by providing software virtualization abstraction over this design. There are extensive studies in literature that demonstrate the issues with such virtualization mechanisms [2, 3, 4, 5, 6, 7]. As a consequence of which, PCI-SIG (Peripheral

Component Interconnect Special Interest Group) on I/O virtualization formulated constructs for providing hardware virtualization support for I/O device access on virtualized platforms [8]. However, I/O Virtualization (IOV) constructs are beneficial for highly I/O intensive workloads wherein performance is critical. Also, IOV platforms impose VM migration constrains. In large Cloud data-centers where a varied mix of different types of resource and performance requirements manifest, it is necessary to look at commodity platforms and see if one can leverage performance guarantees without changing the commodity nature of the solution. It is in this context that we focus mainly on I/O intensive applications contending for common disk resources on a single server in Cloud storage that may suffer unpredictable delays due to multi-tenancy. Virtualization techniques are anticipated to incur processing overheads due to additional layers of software that provide abstraction of hardware resources such as disks. In order to achieve fine-grained control for concurrent disk access scenarios, software mechanisms for allocation of resources such as disk scheduling are commonly employed. In most Cloud storage setups, allocation of disk bandwidth to concurrently running I/O applications is usually done on fair sharing basis with best effort mechanisms. The role of intelligent scheduling techniques for allocation of disk resources, therefore, becomes very important for providing performance guarantees for I/O applications with diverse requirements. The present work proposes novel scheduling schemes that can derive optimal application performance along with maximum resource utilization without requiring any changes to existing hardware architectures.

This chapter introduces the problem statement through illustration of the performance issues that are common for Cloud storage and discusses the approach taken by this work to solve these issues. Section 1.1 discusses the scenario of multiple heterogeneous I/O intensive applications sharing disk resources and techniques for satisfying performance requirements. The role of scheduling techniques for allocation of resources to the applications has been emphasized. Section 1.2 discusses the trade-off between application performance and workload consolidation in typical Cloud setups. The need for differentiated services for I/O applications having different latency requirements has been motivated through experimental validations in Section 1.3. Section 1.4 describes the objectives of this thesis in terms of proposing novel scheduling frameworks for achieving an optimal application performance while maximizing resource utilization. The organization of the thesis is presented in Section 1.5 and the chapter is concluded with a summary in Section 1.6.

1.1 QoS for I/O Applications in Cloud

Cloud computing caters to a diverse range of applications with different functionality and resource requirements such as social networking, enterprise workloads and research computations

[9]. Most applications executing on the Cloud essentially have some disk read or write operations and there are many common applications that need extensive I/O for their functionality. Typical examples are file server workloads, multimedia streaming applications [10], database management systems [11] as well as scientific jobs running for long duration of time [12, 13].

Considering the large number of I/O applications being hosted on the Cloud platform, it may be the case that applications having different kinds of disk I/O workload patterns (e.g. random or sequential read/write requests) and varied performance requirements in terms of response time and throughput are executing in VMs that are co-located on the same host. Typical I/O intensive applications have different latency requirements, request-response workloads [14] and applications like database transactions are interactive and it is imperative for them to complete within desired time limit in order to get useful output [15]. On the other hand, I/O applications like logging activities, research jobs are delay tolerant and do not have strict deadlines for completion.

In order to satisfy the QoS of such I/O applications running concurrently, it is important to capture the latency characteristics of all the applications in terms of the users' expectations. Subsequently, the latency metrics are translated in terms of the desired resource allocation metrics like disk bandwidth to satisfy the performance requirements of multiple contending I/O requests. Since current architectures are not designed for facilitating shared bandwidth access in an isolated manner, disk scheduling is the most widely used method to assign resources for achieving varied I/O performance expectations.

However, the common disk scheduling policies employed in practice are unable to distinguish among the requests from various applications and are oblivious to their performance expectations. If the resource allocation is fair and equal for all the concurrent applications irrespective of their I/O performance requirements, it may cause undesirable delays for latency sensitive applications. Therefore, the disk scheduling policies must be made cognizant of the latency requirements of the applications while allocating bandwidth resources. The difference in latency sensitivity of the various I/O applications can be utilized for achieving desired performance for the set of applications placed on the same server by providing differentiated service allocation. The scheduling and resource allocation policies of the host servers, therefore, assume an important role in ensuring QoS guarantees for the applications running in virtualized environments [16]. For satisfying QoS, differentiated services must be provided to various applications hosted on the physical servers based on their latency characteristics translated into corresponding bandwidth requirements. This thesis proposes the design of a latency aware disk scheduler *PriDyn* which enables the same for the Cloud servers. The details of *PriDyn* scheduler will be discussed in Chapter 3.

1.2 Performance and Consolidation Issues

Cloud computing offers better utilization of physical resources through a shared usage model that can enable substantial energy savings [17]. The underlying hardware resources such as CPU, memory, disk, network bandwidth and others are multiplexed between all the concurrent applications that are hosted on the same server in this model using virtualization methods. For I/O intensive applications, large variations in performance may be observed due to the shared usage, primarily depending upon the type of storage [18] and the degree of multi-tenancy on the server i.e. the number of VMs co-located on the same host, sharing the hardware [13]. An end user running applications on the Cloud may expect to get dedicated access to the physical resources like disk and network in an ideal scenario, but due to sharing the applications only get a proportional share of the bandwidth. This may adversely affect the performance of the applications causing SLA violations, which in turn might lead to undesirable penalties for the Cloud provider. On the other hand, if resources are over allocated to the applications in order to minimize interference and ensure performance, Cloud provider may not be able to achieve good workload consolidation for the servers leading to poor utilization and thereby energy inefficiency. Large scale data-centers need substantial power resources for their continuous operation incurring large energy costs as well increasing carbon emissions. Reports [19] indicate that data-centers and servers in the United States consumed 61 billion kilowatt-hours (kWh) in 2006 (1.5 percent of total U.S. electricity consumption) for a total electricity cost of about 4.5 billion dollars. In 2013, U.S. data-centers consumption was estimated at 91 billion kWh which is projected to increase to roughly 140 billion kWh annually by 2020, causing the emission of nearly 150 million metric tons of carbon pollution annually [20]. Given the ecological and economic impact, it is important to focus on developing energy efficient paradigms for cloud computing. Efficient VM scheduling techniques that place the active VMs on minimum number of physical servers while transforming others to low-power state can enable significant power savings through server consolidation [21].

The dichotomy between application performance and server consolidation becomes highly relevant in setups with high degree of multi-tenancy where multiple VMs accessing the shared resources at the same time may result in highly unpredictable performance [22, 23]. It is pertinent that such issues are addressed so that users get guaranteed performance for their services hosted on the Cloud and at the same time good resource utilization of the system is also maintained.

Usually, in Cloud data-centers, a generic placement policy decides the allocation of physical hosts to different VMs based on system boundary conditions (i.e. hardware capacity) and resource requirements of the VMs [24, 25]. However, there is a need for intelligent meta-scheduling techniques for VM placement to obtain optimal workload combinations on the servers that can achieve the desired balance between application performance and server consolidation. For example, if there is an optimum combination of latency sensitive and other delay tolerant I/O applications, it is possible to attain good performance for all applications while maximizing the utilization of disk resources at the same time. On the other hand, if all the I/O applications co-located on a server are latency sensitive, there will essentially be degradation of performance. To tackle these issues, we have proposed the design of meta-scheduling framework *PCOS* to achieve application performance while ensuring workload consolidation for the Cloud data-centers. The details of *PCOS* are discussed in Chapter 5.

1.3 Multi-tenancy Issues for Disk I/O Performance

This section motivates the need for smarter scheduling techniques by discussing the most pertinent issues in shared Cloud storage environments. Detailed experimental analysis was performed to demonstrate the degradation of disk performance in multi-tenancy environments. The experiments were conducted on a private cloud running OpenNebula3.4.1 cloud framework with KVM [26, 27] as the hypervisor to set up VMs. The host server has 12-core AMD Opteron 2.4 GHz Processor, 16GB RAM and 1TB SATA2 disk(7200 rpm,16MB cache). A dedicated partition on the disk was used for all the I/O experiments to minimize interference. All the VMs were configured identically with 20GB virtual disk space, 1GB RAM and 1 vCPU pinned to an exclusive core so as to avoid the effect of CPU scheduling policies.

To model the I/O patterns of real world applications, the *IOzone* file system benchmark tool [28] was used. *IOzone* is a popular and versatile tool for file and I/O benchmarking offering a wide variety of I/O access patterns. We configured our tests to eliminate cache and buffer effects to get native read/write performance for the disk. The Linux *dd* command was used to model simple block read and write operations.

Several case studies were performed to study the disk I/O behavior for Cloud setups and factors affecting the performance. We analysed the functionality of the hypervisor in terms of facilitating I/O access to the virtual machines running on a single host with help of various experiments. The hypervisor causes a performance overhead for disk I/O which was found to increase with increasing number of VMs as shown in Section 1.3.1. In Section 1.3.2, we examine the relation between disk I/O performance and the degree of multi-tenancy on a single host. In addition, several tests were conducted to understand and illustrate the role of host disk scheduler for achieving desired performance for multiple concurrent applications as discussed in Section 1.3.3.

1.3.1 Role of Hypervisor

The hypervisor or virtual machine manager is a software that enables virtualization by the abstraction of hardware resources to the guest operating system. It performs the essential task of managing device access for the virtual machines. For multiple VMs demanding concurrent disk access, the hypervisor is the single point of resource control and conflict resolution for all the requests. For this reason, the hypervisor becomes a bottleneck and induces substantial overheads for the I/O operations. In our setup, we have employed KVM or Kernel-based Virtual Machine which is a full virtualization hypervisor based on hardware-assisted virtualization wherein the Linux kernel is provided hypervisor capabilities by addition of new modules. KVM is comprised of a set of loadable kernel modules, *kvm.ko* that provides the core virtualization infrastructure and a processor specific module, *kvm_intel.ko* or *kvm_amd.ko*, and also a user space *qemu-kvm* process. *Qemu-kvm* is a modified version of *qemu* [29] which is used for hardware emulation for the virtual machines. There is one such *qemu-kvm* process associated with every VM. A separate qemu thread is assigned to each virtual CPU (vCPU) allocated to a particular VM. The *qemu-kvm* process is responsible for handling all the I/O requests from the VMs through a separate *iothread*. Each VM is a process on the host and it is scheduled as usual by the host CPU scheduler.

In KVM, a third operating system mode called the guest mode is introduced apart from the traditional user and kernel mode. The VMs run in guest mode until a privileged operation like an I/O request is issued. To handle this, a trap is generated and there is a switch to the kernel mode from the guest mode. It is examined if the reason of the switch is an impending I/O operation in which case the control is passed to the corresponding *qemu-kvm* process in user space to perform I/O on behalf of the VM. Then the I/O request is completed in the same way using the system calls as done for other normal processes running within the Linux operating system.

Due to the mode switches associated with every I/O request, there is an inherent virtualization overhead associated with the I/O operations executed from inside a VM. Further, this overhead increases with increasing number of VMs deployed on a single virtualized host. To illustrate this, we executed a simple *IOzone* block write command simultaneously on multiple VMs and tested with increasing number of VMs. For each test, we measured the percentage of various KVM components i.e. *kvm_amd* and *kvm* modules and the *qemu-kvm* process, out of total samples collected with help of *OProfile* tool [30] and the results are shown in Figure 1.1. It is observed that the percentage contribution of the KVM components increases as the number of VMs running I/O operations increases. The percentage component of *kvm_amd* module

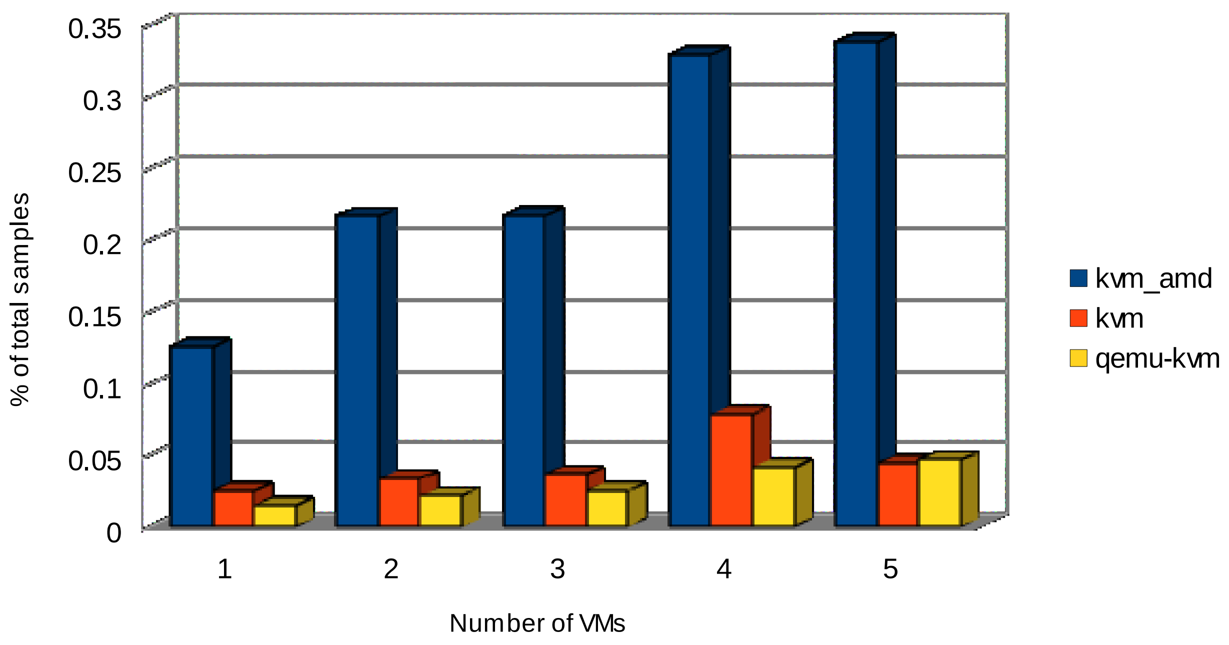


Figure 1.1: Increase in hypervisor overhead with multi-tenancy

almost doubles when moving from a single VM to two concurrently running VMs. Similar increases are noticed for other KVM components as well with increasing number of VMs, thus confirming that the hypervisor overheads depend on the degree of multi-tenancy. It is to be noted that such overheads are not present when I/O requests are executed on a non-virtualized setup. This study clearly indicates that the total device bandwidth capacity for Cloud servers is restricted due to virtualization overheads. Therefore, consolidation of virtual machines on a physical host in virtualized environments must necessarily take into account these overheads and closely match the VM requirements with actual total capacity before attempting to spawn new VMs on the server.

1.3.2 Effect of Multi-tenancy on Disk Performance

To characterize and understand the effect of multi-tenancy on application performance, we examine the variation in disk throughput with different number of concurrent VMs against varying record sizes for write operations. I/O record sizes affect the net throughput achievable as indicated in Figure 1.2 and Figure 1.3. Figure 1.2 indicates the average disk throughput obtained for I/O operations in case of different number of VMs in MB/sec. Disk throughput is higher for larger record sizes in all cases. It is observed that there is a considerable drop in disk throughput (from 90 MB/s to 60 MB/s) in a virtualized environment as compared to the non-virtualized case where the I/O operation is executed directly on the host, without hypervisor intervention and competing tenants. This clearly indicates that the hypervisor causes an overhead for the I/O applications running inside VMs. This overhead increases with increasing number of VMs deployed on a single virtualized host. Further, it can be observed from Figure 1.2 that the average throughput value per I/O operation decreases as we increase the number of VMs on which file write operations were concurrently running. Such a trend is observed for I/O requests of all record sizes with sharper decrease in throughput for larger record sizes.

Figure 1.3 shows the cumulative disk throughput obtained for all the I/O operations that are executing concurrently inside different VMs on the host. For all the block sizes, it is observed that total disk throughput value deceases with increase in number of VMs (40 MB/s for 3 VMs) indicating loss of total disk bandwidth in case of multiple concurrent I/O processes. This behavior is attributed to the interference at disk level where I/O requests from different VMs contend with each other for disk access. The marked non-linear decrease in disk performance observed with increase in number of VMs poses scalability issues putting a limit on the degree of multi-tenancy for I/O.

The variation in disk throughput and the associated latency values for concurrent I/O applications having different characteristics is further discussed next with reference to disk scheduling of the I/O requests.

1.3.3 Role of Disk Scheduler

In virtualization setups, every virtual machine is assigned physical disk space on the host which is exposed as a virtual disk inside the VM. Since all virtual machines are represented as normal user level processes on the host, all I/O requests from the virtual machines (handled by qemu) are dispatched to the actual disk by the disk scheduler of the host. The host disk scheduler cannot distinguish between the I/O requests coming simultaneously from different VMs or from the host itself. Due to limited information about the I/O latency requirements of the requests, the disk scheduler is unable to provide differentiated performance to the I/O applications of the VMs and all the applications suffer delays due to multi-tenancy regardless of their latency requirements. In order to model the I/O workloads that are typically hosted on cloud environments, concurrent bulk read and write operations were executed on different VMs in various combinations.

Figure 1.4 shows the average disk throughput values obtained for various cases. It was observed that a single read or write operation having exclusive disk access achieved the highest

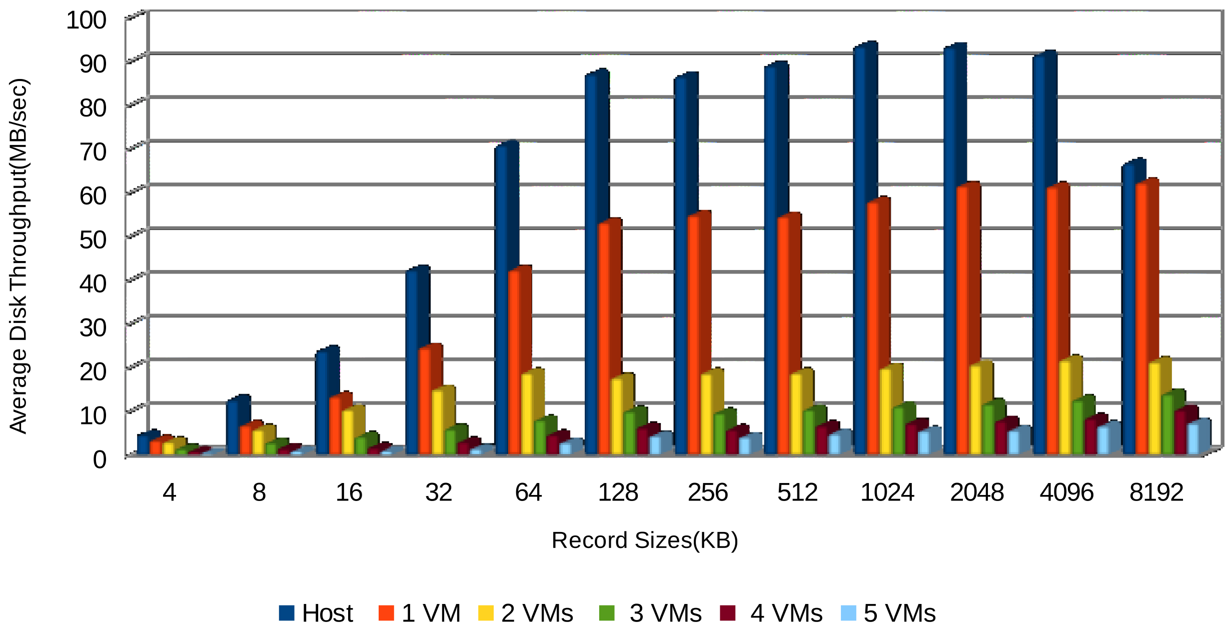


Figure 1.2: Average disk throughput with varying record sizes for different number of VMs.

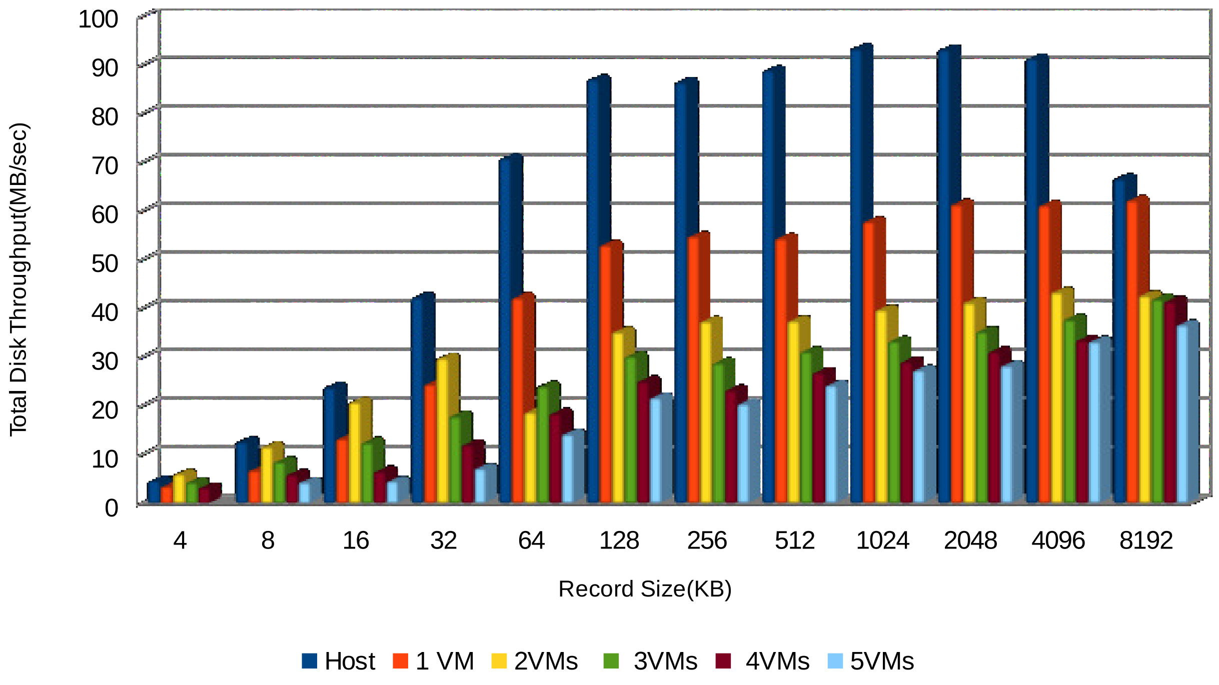


Figure 1.3: Cumulative disk throughput with varying record sizes for different number of VMs.

disk throughput value. If a read process is executed simultaneously with an ongoing write process on another VM, there is a significant drop in disk bandwidth for both the processes. Similar observations were made for other combinations of read and write operations as indicated in the figure. It is clearly seen that the throughput value decreases equally for all the applications irrespective of their workload pattern or service requirements.

The latency or the response time for an I/O operation is closely linked with the disk throughput value. Higher disk throughput for an operation implies lower latency value. Figure 1.5 indicates the latency values that were observed for the same combinations of read and write operations. It is observed that the latency value for all the I/O operations increases as the number of concurrent I/O operations increases independent of their type. Thus, we can conclude that it is essential to make the disk scheduler in the host aware of the I/O characteristics of applications running in the VMs so that desired performance can be achieved.

To summarize, following are some of the pertinent issues regarding disk I/O performance that were illustrated through our experiments :

- The hypervisor causes an overhead for all I/O operations from the VMs which increases with rise in the number of VMs. (Section 1.3.1)
- The cumulative disk throughput decreases as the number of I/O operations running concurrently inside VMs that share the physical disk increases. (Section 1.3.2)
- All concurrent I/O applications suffer from performance degradation as the host disk scheduler does not consider variations in performance requirements. (Section 1.3.3)

1.4 Objectives of the Thesis

The previous section has discussed the most crucial issues for disk I/O performance for Cloud storage setups. It has been shown with various experiments that the disk I/O schedulers by default can only achieve best effort services to all the I/O applications irrespective of their urgency of operation. In order to satisfy the QoS requirements of a wide ranging I/O applications typically running concurrently on the Cloud hosts, we need to provide differentiated services to the applications based on their latency requirements.

In this work, we discuss our proposed novel disk scheduling framework *PriDyn (DYNamic PRIority)* which provides differentiated services to various I/O applications co-located on a single host based on their latency attributes and desired performance. It is assumed that Cloud users specify the performance requirements of the I/O applications in terms of the data size (for bulk reads/writes) or request size (for transactional applications) and the desired

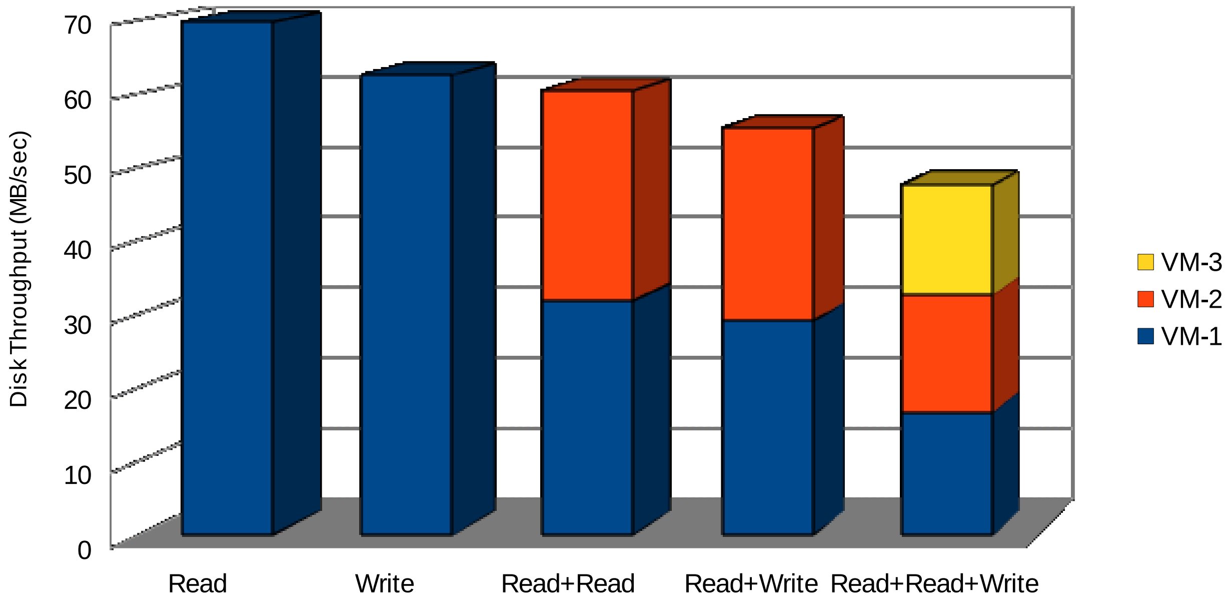


Figure 1.4: Disk throughput for different combinations of I/O operations

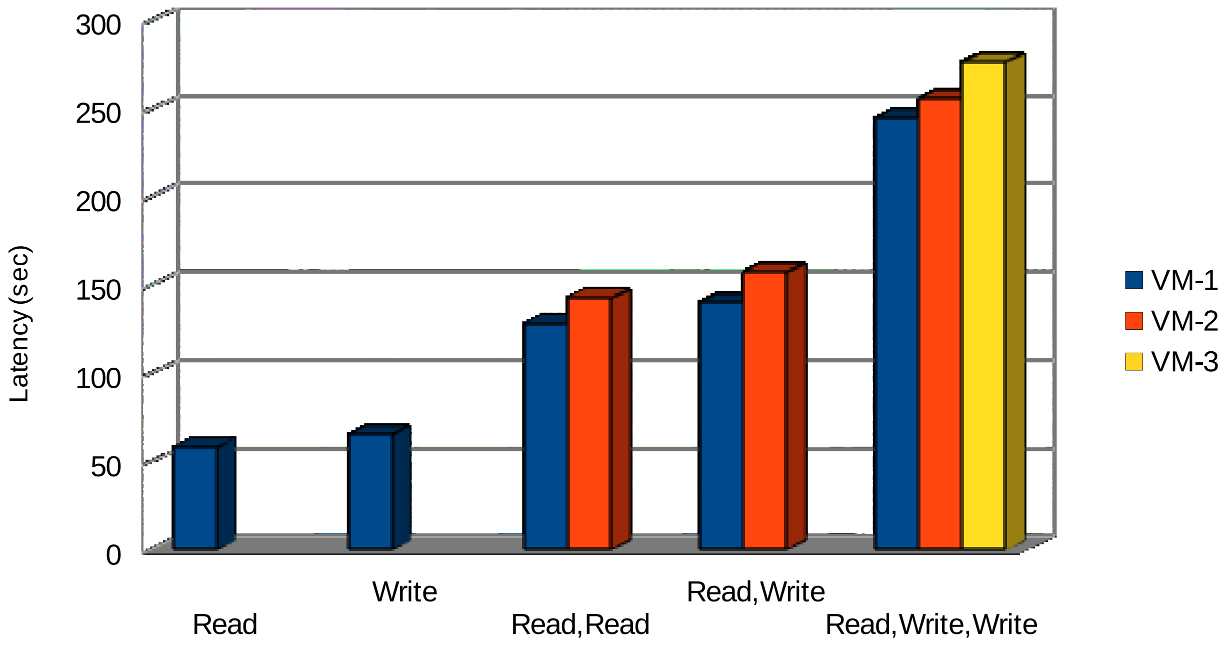


Figure 1.5: Service latency for different combinations of I/O operations.

deadline i.e. the maximum time by which the application is required to finish execution. To achieve differentiation in I/O performance, the framework employs a scheduling algorithm which dynamically computes latency estimates for all concurrent I/O applications for a given system state. Based on these, an appropriate priority assignment for the applications is determined which will be taken into consideration by the underlying disk scheduler at the host while scheduling the I/O applications on the physical disk. This will essentially address the concerns about the scheduler being indifferent to the varied performance requirements of the applications as identified in Section 1.3.3. Thus, this framework describes a performance-driven latency-aware application scheduler on top of the actual disk scheduler on the host in a virtualization environment. It ensures that critical applications do not suffer latency in shared environments, enabling significant improvement in I/O performance and give QoS guarantees.

Moreover, it is essential to follow optimal application scheduling strategies for the servers at the data-center level working in conjunction with the disk I/O schedulers at all individual servers. The effects of different scheduling approaches for I/O workloads with diverse latency requirements on the same server have been illustrated and analyzed later in Chapter 5 . Desired performance for all the I/O applications executing on a single server can be enabled by intelligent meta-scheduling techniques and admission control mechanisms. With the correct combination of I/O applications on a server having varied latency requirements, the *PriDyn* scheduler can achieve desired performance objectives for all the applications.

Therefore, the *PriDyn* framework was extended and a novel admission control and meta-scheduling framework named ***P****rescient* ***C****loud I/****O*** ***S****cheduler* (*PCOS*) was designed for obtaining good I/O workload combinations for Cloud systems. *PCOS* aims to maximize the utilization of disk resources without adversely affecting the performance of the applications scheduled on the systems. This scheduler follows a conservative methodology in which migrations of I/O applications among the systems is strictly avoided. By anticipating the performance of the systems running multiple I/O applications, *PCOS* can prevent the scheduling of undesirable workloads on them. The *PCOS* framework includes the *PriDyn* scheduler as an important component and utilizes the dynamic disk resource allocation capabilities of *PriDyn* for meeting its goals. With this framework, we can achieve an improvement in disk I/O performance even after taking into consideration various hypervisor and disk overheads as discussed in Section 1.3.2 and 1.3.3. Therefore, *PriDyn* disk scheduler and *PCOS* meta-scheduling framework together are capable of addressing the issues identified for disk I/O performance in Cloud storage setups. We prove through modeling and validations on real I/O workload traces that with our approach, storage resources can be allocated in an optimal manner based on the performance requirements of the applications in Cloud data-centers.

1.5 Organization of the Thesis

The remainder of the thesis is organized as follows.

Chapter 2 discusses the background and related work previously done that is relevant in the current context. We state the existing scheduling techniques and emphasize how the proposed framework better addresses present concerns.

Chapter 3 introduces the design and functionality of the *PriDyn* disk scheduling framework. The implementation details of the framework which enables it to take into consideration varied performance requirements of the applications have been discussed. This scheduler achieves service differentiation for allocation of disk resources to concurrent applications on a single host.

Chapter 4 provides the proof of concept of the proposed *PriDyn* framework with help of various experiments. Real world traces were used for validating the functionality and utility of *PriDyn* for actual Cloud storage scenarios. Results show that *PriDyn* achieves significant performance enhancements.

Chapter 5 illustrates the need for meta-scheduling of workloads in Cloud data-center through various case studies. It describes the *PCOS* meta-scheduling framework which intelligently selects an optimal combination of workloads for the Cloud servers to enable good server consolidation while maintaining desired QoS. This chapter presents design and implementation details along with the experimental validation for *PCOS* framework indicating that *PCOS* achieves better workload consolidation for servers and maximizing disk utilization without any performance degradation.

Finally, Chapter 6 concludes the thesis and discusses future work.

1.6 Summary

This chapter provides the necessary background for understanding the rest of the thesis. The most pertinent issues for the optimal performance of I/O applications on Cloud storage setups were discussed and illustrated with the help of experimental results. The chapter motivates the need for differentiated services and proper meta-scheduling strategies for Cloud data-centers, thus giving an overview of the approach that has been adopted by this thesis to tackle the multi-tenancy issues and ensure guaranteed services for Cloud applications. In the next chapter, we discuss the background and explore available literature related to our work in detail in order to elicit the advantages of our approach and place our contributions in the correct context in the field of virtualization and Cloud systems.

Chapter 2

Background and Related Work

Performance issues associated with virtualization techniques have been studied previously [31]. These challenges become more relevant for I/O operations as the limited I/O resources become a bottleneck while trying to attain application level QoS. I/O performance bottlenecks for data intensive workloads with commercial publicly available as well private Cloud setups have been demonstrated in [32], motivating the need for continued efforts for better systems and practices in I/O virtualization.

In this work, we emphasize that QoS guarantees can be provided for the applications running on every server through novel disk scheduling methods. The characteristics of various co-hosted applications must be identified and disk resources should be allocated to the them dynamically according to their requirements. In addition, smart meta-scheduling techniques should accompany server-level scheduling to ensure proper scheduling of VMs on the server at the data-center level. Through efficient scheduling practices, better consolidation of workloads can be attained which can reduce power consumption of the data-centers.

We discuss existing literature related to both of these aspects separately.

2.1 QoS for co-tenant VMs on a Server

Efforts to provide QoS for disk I/O have been made in earlier works such as [33, 34] but these attempts were based on classical operating systems and they do not consider the complexities of resource allocation involved in virtualized Cloud storage environments. In [35], authors have studied such complexities associated with virtualized Cloud setups due to the abstraction by the hypervisor layer and proposed a new disk I/O model *HypeGear* for the coordination and management of multiple guests on the hypervisor. For concurrent I/O requests from multiple virtual machines, which lead to the disk becoming a point of contention, the proposed model aims to achieve disk I/O optimization with help of system-wide and device-specific information. However, this model uses caching in host OS to improve disk I/O bottleneck and does not focus on scheduling techniques to alleviate multi-tenancy issues.

The effect of disk scheduling on the performance of I/O applications in virtual setups has been studied by Boutcher et al. in [36]. By comparing the fairness and throughput metrics achieved with different Linux I/O schedulers on a virtualized system, the paper shows that for obtaining better overall performance, the choice of scheduler should be based on the application workload on a virtual machine. In [37], Kesavan et al. have discussed in detail the role of hypervisor level scheduler in ensuring good performance for the I/O application executing on the VMs. Similar studies on performance impact of the VMM scheduler on I/O intensive applications had been undertaken by Ongaro et al. [38] and Cherkasova et al. [39] for the Xen virtualization platform [40]. In [38], the authors have examined optimizations to Xen's Credit scheduler for improved I/O performance and emphasised the need for further enhancements. [39] examines the effect of the choice of scheduler and its parameters on application performance and identifies the challenges in allocation of CPU resources to applications as per requirements. The current work focuses on achieving performance improvement through scheduling of disk I/O applications particularly with regard to their latency characteristics.

Much literature is available for providing fairness and performance isolation to I/O applications in virtual environments. In [41], Lin et al. have analyzed performance interference issues when different types of random and sequential I/O workloads are co-located on single host and proposed a fair sharing framework which uses replication for reducing the interference. Similarly, in [42], Kim et al. have studied the reasons for fairness-violation and I/O performance degradation in virtual machines. The paper proposes a virtualization-aware fine-grained I/O analysis framework which keeps track of scheduling information and devises techniques for enforcing fairness. In [43], the authors have proposed a virtual I/O scheduler which claims to ensure fair and isolated performance by fair allocation of disk time to the applications. [44, 45] proposed similar ideas for ensuring performance isolation for the I/O workloads of different virtual machines sharing storage. Works like [46, 47, 48, 49, 50] have focused primarily on providing proportional resource allocation with the goal of achieving fairness in I/O performance. In [51], the authors have proposed a proportional share scheduler which used separate I/O queues for each VM to provide QoS.

Recently, in [52], Sfakianakis et al. have identified that fairness is not an appropriate optimization metric for shared servers catering to different workloads. They have proposed *Vanguard* which enforces I/O path isolation for interfering workloads on consolidated servers to achieve higher server utilization. In *DVT*[53], latency smoothing techniques were used to reduce the rate of change of service latency and get proportionality and isolation across VMs.

Our work is different from aforementioned approaches as we have focused on providing differentiated services to I/O applications taking into consideration the variability in service requirements for different applications in shared Cloud storage. Unlike previous work, the *PriDyn* framework proposed in this paper aims to meet service time deadlines by dynamically allocating disk resources to I/O applications based on their latency requirements. It manages the distribution of disk resources to the applications at high granularity to efficiently allocate disk bandwidth among applications according to fluctuating needs. Thus, *PriDyn* has been designed as a latency aware disk scheduler to achieve QoS guarantees for the applications.

2.2 VM Placement and Scheduling in Data-centers

In order to realize the full benefits of disk scheduling at the servers through the framework, complementary VM placement strategies are necessary at the data-center level to ensure the scheduling of optimal workloads on the servers. In [31], Koh et al. have attempted to analyse the performance interference for virtualized workloads and make predictions for the performance of different combination of workloads based on their characteristics. We have taken similar approach in this work and taken it a step further by aiming to achieve server consolidation through efficient resource utilization along with application performance.

Attempts to provide guaranteed services to application running inside co-hosted VMs by assigning excess resources leads to resource wastage for the providers in a Cloud setup. As such, there have been many attempts to optimize resource utilization through efficient VM management and scheduling techniques for data-centers. The available work in literature on scheduling of I/O applications in virtualized storage setups can be discussed in two categories - those that aim to achieve good performance for I/O applications through scheduling strategies and others where the primary goal is consolidation of data-center with the help of load balancing techniques. The framework proposed in this work is unique as it aims to combine the techniques for I/O application scheduling with server consolidation objectives to achieve good resource utlization but without making any compromises on application performance at the same time.

2.2.1 VM Scheduling for Application Performance

In this category, Casale et al. [54] discuss the impact of sharing of virtual disk resources by predicting performance in multi-tenant environments. [55, 56, 22] have performed I/O workload characterizations to anticipate performance and enable better I/O scheduling. We have taken a similar approach but unlike previous work we have focused on optimal workload scheduling based on differentiated I/O service requirements. We have employed admission control techniques to obtain suitable workload combinations that ensure application performance along with maximum resource utilization. Such techniques have been previously used in other works like [57, 58] which enable dynamic provisioning and scheduling of tasks but they do not consider the complexities involved with handling I/O resources.

In [59], Delimitrou et al. propose *Paragon*, an interference aware scheduler which classifies workloads as per their requirements to find a suitable server for scheduling. Similar to our work, they aim to achieve both application performance and efficient resource utilization by finding the least interfering placement of VMs on hosts. But this work also discusses computing resources and is not relevant to I/O intensive workloads.

2.2.2 VM Placement for Consolidation

Consolidation of resources through load balancing and placement strategies has been extensively discussed by Gulati et al. [60, 61, 62]. Earlier studies like [63, 64, 65] were concerned primarily with storage configuration and optimization techniques. In [60], authors have proposed an automated tool for storage management through I/O load balancing techniques.

DeepDive [66] aims to ensure performance for IaaS Cloud environments by identifying and managing interference among VMs on a server and suggesting suitable migrations to alleviate the issue. Closer to our work, [61] discusses the need for workload characterization and device modeling for automated I/O placement with help of VM migrations. The aforementioned works achieve consolidation mainly through migrations whereas we have designed our framework to minimize the costly task of I/O application migrations while realizing good workload combinations for data-center consolidation.

Q-Cloud [67] has similar motivation as that of our work, it attempts to achieve performance isolation for co-hosted VMs as well as increase resource utilization. According to user defined service levels, excess resources are set aside with each server to handle dynamic surges in resource demand by the applications and meet QoS. The decision regarding excess resources and placement of VMs on the servers are made with help of predictions. However, the framework is unable to perform in the absence of sufficient extra resources which causes resource wastage. In contrast, in our work we try to achieve guaranteed services through the dynamic allocation and reallocation of available limited I/O resources. Also, *PCOS* meta-scheduler performs intelligent placement of VMs without requiring the need for any extra resources on the servers unlike *Q-Cloud*.

2.3 Summary

This chapter discussed previous literature related to the domain of I/O performance considerations in virtual Cloud setups. Through the proposed framework in this thesis, we have tried to address the concerns regarding latency aware differentiated I/O performance QoS for virtualized applications which has not been explored till date to the best of our knowledge. Also, unlike previous works, we have attempted to enable better utilization of hardware resources in Cloud data-centers while also ensuring performance guarantees at the same time. In the next chapter, we discuss in detail the *PriDyn* scheduling framework for providing latency specific I/O services to the applications.

Chapter 3

PriDyn Scheduler

Cloud systems host a wide range of heterogeneous I/O intensive applications as discussed in Chapter 1. According to the functional use cases of the applications hosted inside VMs, there may be varied latency bounds which result in different bandwidth requirements for different applications. For instance, an online video streaming application like Netflix, which uses Amazon cloud services extensively for its operations [10], will have strict I/O latency bounds for acceptable performance. Similarly, an interactive transaction-based application, such as a database system will have a desired response time limit for each user query. On the other hand, high-throughput computing tasks or large scientific applications such as n-body simulations, molecular dynamics problems have long execution periods and do not have strict deadlines for completion.

The disk scheduling techniques commonly employed for I/O applications in prevalent technologies are not suitable for such dynamic environments as they are unaware of the characteristics of the various I/O applications generating disk I/O requests. This is especially true for Cloud setups since on multi-tenanted hosts, the local storage allocated and visible to a VM is a software abstraction of the physical disk of the host. Due to this, the I/O request characteristics that are visible to the VM are opaque to the actual disk scheduler that resides inside the hypervisor at the host. Therefore, most of the popular mechanisms for sharing disk storage are designed for weighted fair-sharing that typically uses I/O demand from each VM. However, such mechanisms are unable to ensure performance guarantees since the resource availability depends on the degree of multi-tenancy. For guaranteed performance, the application would require exclusive access to the disk resources. But such a policy would affect the consolidation ratios for I/O workloads on Cloud setups and impose limitations for both users as well as the providers. In order to alleviate this issue, we have proposed the *PriDyn* scheduler which is a novel disk scheduling framework to achieve performance-specific QoS for Cloud storage. By taking cognizance of the functional characteristics of the I/O applications, *PriDyn* aims to fulfill application performance goals specified as QoS requirement without unduly loosing out on the host's resource utilization efficiency. It translates user-level requirements to system level specifications such that the allocation of disk resources to the applications can be fine-grained and dynamically managed as per the requirement.

This chapter presents the proposed *PriDyn* framework and discusses its features and functionality for enabling QoS based I/O performance in Cloud environments. Section 3.1 gives an overview of the *PriDyn* framework discussing the assumptions and main features. Section 3.2 presents the design considerations and the functionality of the main components Latency Predictor (Section 3.2.1) and Priority Manager (Section 3.2.2). The role of the disk scheduler in the *PriDyn* framework has been explained in Section 3.3. This chapter also presents the algorithm and discusses the implementation details of the *PriDyn* framework in Section 3.4.

3.1 Overview

In Cloud setups, a VM running I/O applications may have the same workload pattern over its lifetime or it may cater to different kinds of I/O workloads at different times depending upon the different types of services offered by the user application. We assume that the placement policies of a data-center take into consideration the configuration of the host machines and decide the placement of VMs on physical hosts according to the system boundary conditions. For instance, a VM is assigned to a host only if the maximum virtualized bandwidth that can be offered by the host satisfies the bandwidth requirement of the VM. Similarly, the number of VMs co-located on the same host will also depend upon the underlying hardware capacity. In addition, multiple VMs running applications with strict latency requirements are not placed on the same host so as to avoid the resulting degradation of performance. Therefore, it is desirable to place VMs having a combination of applications with varying latency requirements on a single host. But in such a scenario, a latency-sensitive application may suffer delays due to interference by other applications contending for disk at the same time. For instance, if a streaming application is executing simultaneously with a long running scientific application, its performance may be adversely affected and it may suffer delayed service time. The proposed *PriDyn* scheduler is a latency aware application scheduler that has been designed to resolve this issue by leveraging the difference in service requirements to satisfy QoS of multiple concurrent applications on a single host.

Figure 3.1 shows the structure and relative placement of the *PriDyn* scheduling framework with respect to the whole setup under consideration. The framework functions as a part of the hypervisor which manages the allocation of physical disk resources of the host to the multiple virtual machines that are being simultaneously executed on it.*PriDyn* is composed of *Latency Predictor* and *Priority Manager* components whose functionality is explained in detail subsequently through Figure 3.2. The proposed framework simultaneously considers system disk utilization as well as application performance in terms of user level agreements to effectively partition the available resources among these VMs. It aims to provide prioritized services to time bound applications in order to meet their performance SLAs while also maintaining acceptable QoS for less critical applications. *PriDyn* achieves this by computing suitable priority values for disk access for I/O applications on the basis of their latency requirements. In this work, we are assuming that though VMs may execute different I/O applications over a period of time, but at any given time, each VM runs a single I/O application on the host. Workload characterization is performed by the user for their I/O applications and attributes such as the total size of the data and the value of deadline associated with each I/O application are provided to the framework as inputs. We assume that these attributes remain same over the period of execution of the application. As an example, for a video streaming application, the user can specify the data size of an I/O request (corresponding to the size of a video file) and the required streaming rate (which can be used to derive the request deadline). Based on this data and the current system state, *PriDyn* computes the latency estimates and the corresponding priority values for all applications after a particular scheduling interval. It uses a feedback-based mechanism to adaptively determine the current application performance on the system in terms of allocated disk bandwidth and recomputes latency estimates at every interval. This scheduling interval can be varied according to the rate of change in the system state such that the framework necessarily recalculates the priorities for all the running applications whenever any new I/O application begins execution or an existing application completes I/O. For e.g., if the applications have short execution times, then the scheduling interval should be small. In our experiments that are demonstrated in Chapter 4, the duration of scheduling interval is 3 seconds. This scheduling interval is distinct from the scheduling interval of the actual disk level scheduler at the host.

A distinguishing feature of the current work is that the priority values are not statically assigned to the applications but are dynamically updated based upon the system state. Static priority assignment policy for I/O applications in a Cloud environment can lead to sub-optimal resource allocation as well as performance limitations. The arrival of critical applications cannot be known in advance in a dynamic Cloud storage setup. Moreover, there may be applications having variable I/O workloads. A static priority scheme cannot modify the priorities of already running applications for reallocation of resources and therefore such a system may not be able to accommodate new applications with higher performance demands on the host. With the

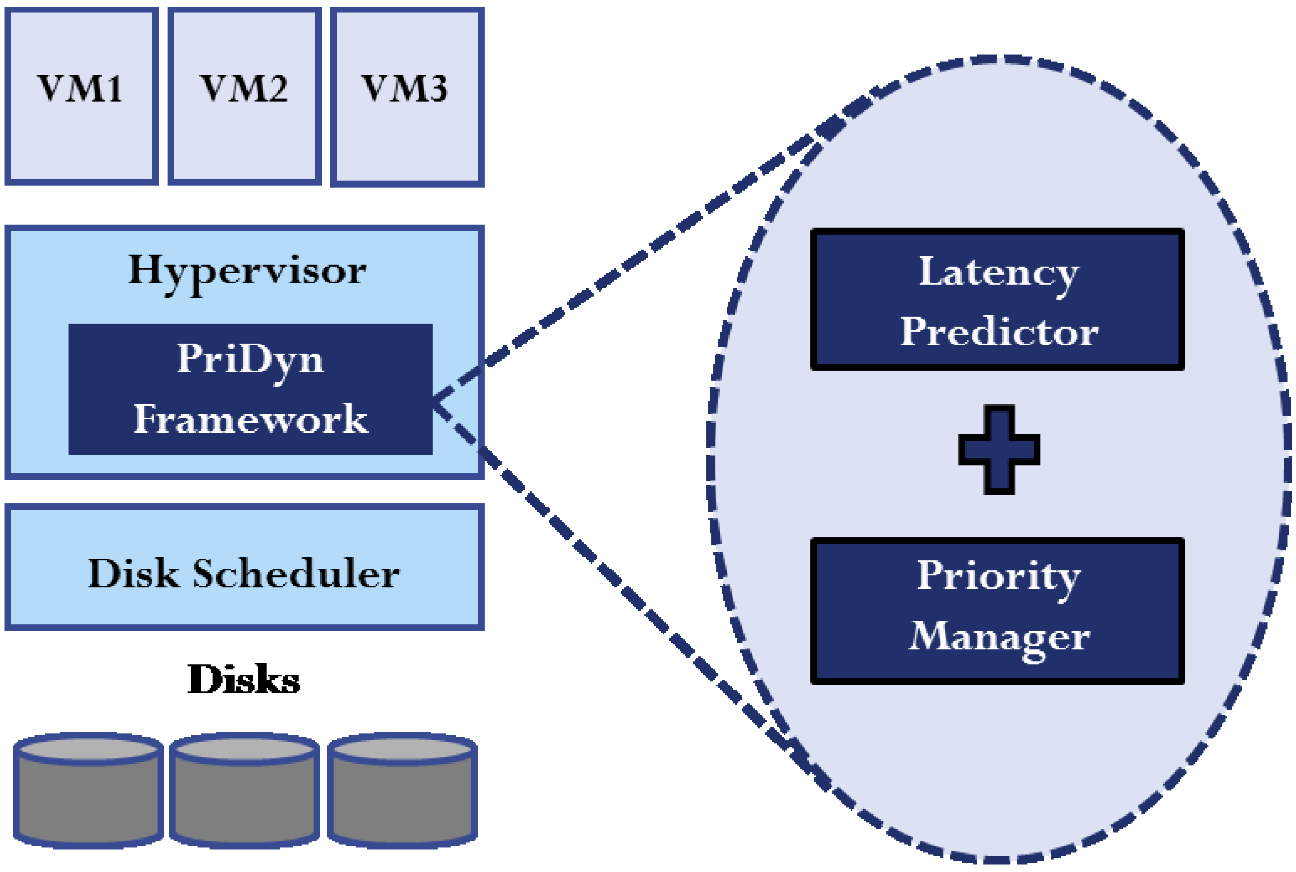


Figure 3.1: PriDyn Framework

dynamic scheduling framework, resource allocation can be optimized and higher performance for critical applications can be achieved. Another key advantage of our framework is that it does not require any changes to the host or guest OS and can be easily integrated into virtualized setups as a hypervisor extension to support QoS.

3.2 Design

Broadly, *PriDyn* consists of two main components- the *Latency Predictor* and the *Priority Manager*. The design of the framework is illustrated in Figure 3.2. The values of deadlines and data sizes belonging to the I/O applications are given to the framework as inputs and the priority values of all the applications are the output of our framework which are passed on to the disk scheduler of the host for assignment to the applications. The functionality and components of the framework is described in detail below.

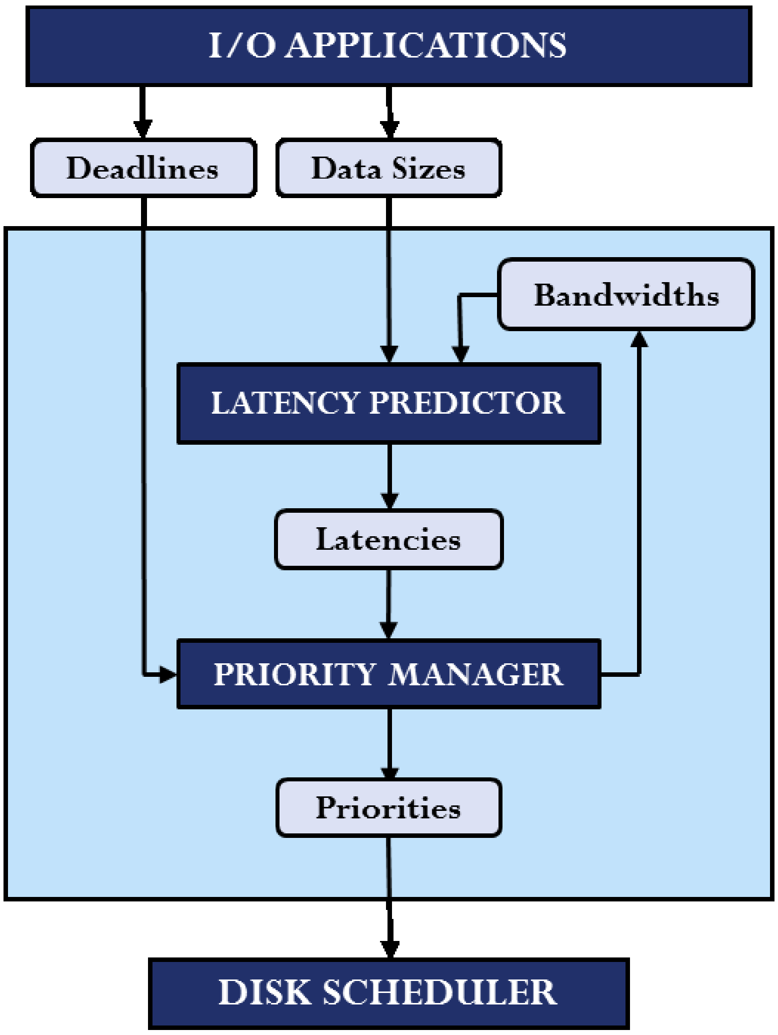


Figure 3.2: Design of PriDyn

3.2.1 Latency Predictor

In order to provide differentiated services to I/O applications, it is essential that the stated application requirements are compared with the actual performance of the application on the system. If a specified deadline for an application is to be met, it is required to obtain an estimate of the finish time for the application (based on present performance parameters i.e. allocated disk bandwidth) so as to decide whether the application needs higher priority services. This is the responsibility of the *Latency Predictor*. It receives as input the total data size (read/write) corresponding to each active I/O application on the host. It obtains the disk bandwidth and calculates the size of the data remaining to be processed for every application. In our setup, the *iotop* tool available in Linux [68] has been employed to obtain the I/O statistics corresponding to the applications running on the system in terms of the disk priority, read and write disk bandwidth available to the processes, as well as a measure of the data read/written by the processes. The framework will record disk statistics using *iotop* at every scheduling cycle. Based on bandwidth value, the *Latency Predictor* calculates an estimated time of completion (latency value) for every application under present system conditions. For each application, the prediction is optimistic in the sense that arrival of a higher priority application can increase the service time and is pessimistic because it does not take into consideration that one or more of the currently executing applications will finish and free up bandwidth which can be allocated to the current application. The expected latency values are computed at the beginning of every scheduling cycle and the set of latency estimates corresponding to all applications are provided to the *Priority Manager*.

3.2.2 Priority Manager

This module enables service differentiation for the I/O applications according to their requirement. In every scheduling cycle, having received an estimate of the latency for every application, the *Priority Manager* compares it with the I/O requirements of the application to determine if the desired deadline is being violated for the current system state. In the event of any violations, the disk priorities for the active applications are dynamically adjusted such that the resultant disk bandwidth allocation for the critical applications satisfies the performance requirement. The priority values will be conveyed to the underlying disk scheduler of the host which will allocate disk resources accordingly. The bandwidth values corresponding to the applications that are obtained after priority assignment are again sent to the *Latency Predictor* as feedback which recomputes the latencies based on them. This procedure is repeated in every scheduling cycle. Since an I/O application running inside a VM is a process on the host, we will refer to the I/O applications as processes when discussing priority assignment at the disk scheduler in later sections of this paper.

The *Priority Manager* guarantees a locally optimal solution in terms of resource allocation and application performance within the system configuration bounds and hardware limitations. In addition to providing service guarantees to critical applications, it also ensures acceptable services for the non-critical applications such that they incur marginal increase in latency values (as shown later in this paper). Also, it identifies the critical applications for which the desired deadline cannot be satisfied under the present system conditions and ensures that other applications do not suffer any performance degradation on the expense of a critical application that is certain to miss its deadline. This will be discussed further while describing the *PriDyn* algorithm in Section 3.4. The IaaS provider can choose to clone or migrate such an application to a different host to satisfy the SLAs. In present work, we focus on the scheduling options for a single host only.

3.3 Disk Scheduler

As discussed earlier, the disk scheduler at host level is oblivious to the I/O characteristics of the applications running inside the VMs. The efficacy of our framework is realized in such a scenario as it calculates the desired priority values for different applications and provides this information to the underlying disk scheduler at the host. To implement prioritized services, we have employed the *Completely Fair Queuing (CFQ)*[69] scheduler at the host in our framework. The CFQ scheduler aims to provide fairness in allocation of disk I/O bandwidth to all processes. It maintains per process queues and dispatches requests to the disk based on the priority values associated with the processes. CFQ has 3 broad priority classes – *real-time*, *best-effort* and *idle* in decreasing order of priorities. All incoming I/O requests are put in per-process queues having designated priorities according to which they are given time slices for disk access. The length of the time slice and the number of requests dispatched per process will depend upon this priority value. Processes with *real-time* priority are given longer time slices during which they have exclusive disk access and therefore get higher disk bandwidth values followed by *best-effort* and lastly *idle*. As long as there are requests in *real-time* queue, requests in *best-effort* and *idle* queues are not given disk time and they might starve. By default, all new processes are assigned *best-effort* priority; however this can be modified with help of *ionice* command or *ioprio_set* system call in Linux. At every scheduling interval, all the I/O applications are re-assigned priority values (which may be same or different from previous scheduling cycle) with the ionice command based on the required priority values calculated by *PriDyn* framework. The performance of the applications will depend upon these priority values. Although other schedulers like deadline are also available in Linux but they are not designed to provide prioritized services with the kind of usage that is discussed in this paper.

3.4 Implementation

The algorithm for implementation of the *PriDyn* scheduler is shown in Figure 3.3. Table 3.1 summarizes the notations used in the algorithm. This algorithm executes at every scheduling cycle of *PriDyn* scheduler and considers all actively running I/O processes. It takes the set of deadline and data size values for all the processes as inputs and provides as output a set of priority values for all the processes. This algorithm is feedback based, it calculates the priorities of the processes at every scheduling cycle based on the disk bandwidth currently allocated to the processes. This disk bandwidth allocated to each process in turn depends on the priority values

Table 3.1: Terminology for PriDyn Algorithm

Attribute	*Notation*	*Description*
Process	$P =< P_1, P_2 \dots P_N >$	The processes currently active on the system, total N.
Deadline	$D =< D_1, D_2 \dots D_N >$	D_i is the time by which the process P_i is required to finish execution (Measured in seconds).
Total Data Size	$R =< R_1, R_2 \dots R_N >$	R_i is the total size of the data to be read/written for process P_i (Measured in KB).
Disk Bandwidth	$B =< B_1, B_2 \dots B_N >$	B_i is the current value of disk bandwidth available to process P_i (Measured in KB/sec).
Latency	$L =< L_1, L_2 \dots L_N >$	L_i is the estimated time that process P_i requires to finish execution based on current bandwidth allocated to it. (Measured in seconds)
Priority	$Pr =< Pr_1, Pr_2 \dots Pr_N >$	Pr_i is the priority value out of {1, 2, 3} assigned to process P_i where 1: real-time($highest$), 2: best effort($default$) and 3: idle($lowest$).
Time Elapsed	$T =< T_1, T_2 \dots T_N >$	T_i is the time elapsed since the process P_i began execution (Measured in seconds).

that had been assigned to the process in the previous scheduling cycle. For a given system state, this algorithm computes the set of priority values which enables optimum resource allocation with respect to performance requirements.

In Figure 3.3, the first part of the algorithm corresponds to the *Latency Predictor* which computes the estimated latency value for every active process based on the current disk bandwidth B and the value of *DataProcessed* (measured with the help of *iotop* tool). For every new process, *DataProcessed* has an initial value of zero and default priority assigned to the process is *best-effort*. The second part of the algorithm corresponds to the *Priority Manager* which tracks all active processes and finds a process, if any, whose deadline is likely to be violated based on the latency estimates and the desired deadline values (Step 6). If all processes are estimated to meet their respective deadlines, the processes continue execution with the default priorities. In case of multiple processes violating deadlines, the process P_i with the earliest deadline is chosen for consideration in the present scheduling cycle. Next, for process P_i, the algorithm finds whether there is any such process P_j whose deadline is later than deadline of process P_i and this deadline is not likely to be violated (Step 8). If there are multiple such processes, the chosen process will be the one for which the difference between deadline and latency value is largest and whose priority is not already the lowest (Step 9). This process is most suitable for being assigned lower priority in the current system state. The following three cases will be considered by the algorithm:

- Case 1 : There exists a process P_j (Step 10) : In this case, the priority of process P_j is decremented (Step 11).

- Case 2: Process P_j is not found, process P_i does not have the highest priority (Step 13) : Here, since there is no suitable process P_j whose priority can be decremented, the priority of the process P_i (whose deadline is required to be met) is incremented to improve the disk bandwidth allocated to it.

- Case 3 : Process P_j is not found, process P_i has the highest priority (Step 15) : In this case it is not possible to achieve the desired deadline of process P_i for any priority assignment scheme under the current system conditions (due to system limitations). Therefore, the algorithm assigns the lowest priority to P_i (Step 16) and restores the priorities of the other active processes to their previous values in case they were changed while attempting to meet deadline for process P_i (Step 17).

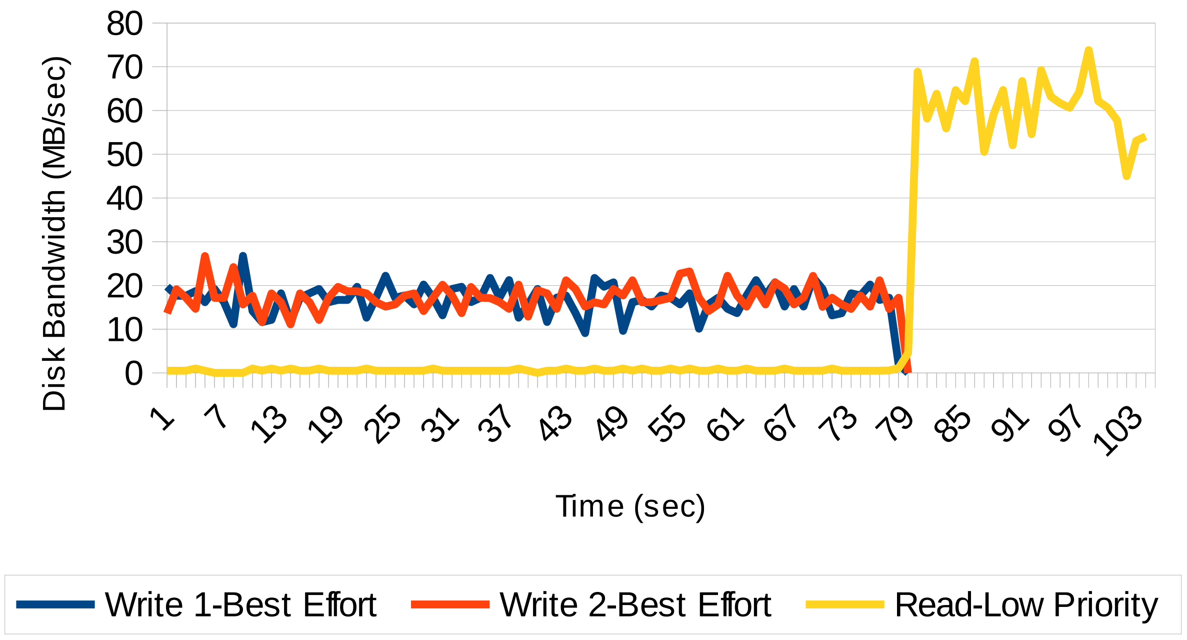
This is done to ensure that other processes do not suffer performance degradation at the expense of one latency sensitive process whose deadline cannot be met for current system state. In such a case, when the other concurrent processes finish execution, process P_i will get dedicated disk access and higher disk bandwidth such that it will finish execution earlier than the latency value estimated by the algorithm.

Figure 3.4a illustrates such a scenario. Two write operations represented by the red and blue lines are running concurrently with a latency sensitive read operation depicted as the yellow line. The read operation has a deadline value which is not achievable for current system state and therefore, *PriDyn* algorithm assigns low priority to read operation as discussed above. It can be seen from the figure that the yellow line stays close to zero value of disk bandwidth while the other two write operations are executing with *best-effort* disk priority values. After the write operations finish execution, the read operation gets exclusive access to disk and thus obtains higher disk bandwidth. This is clearly shown in the figure as the sudden increase in the disk bandwidth value corresponding to the yellow line which represents the read operation. Due to allocation of higher bandwidth value, the read process finishes earlier than the estimated time and is expected to suffer minor degradation in performance.

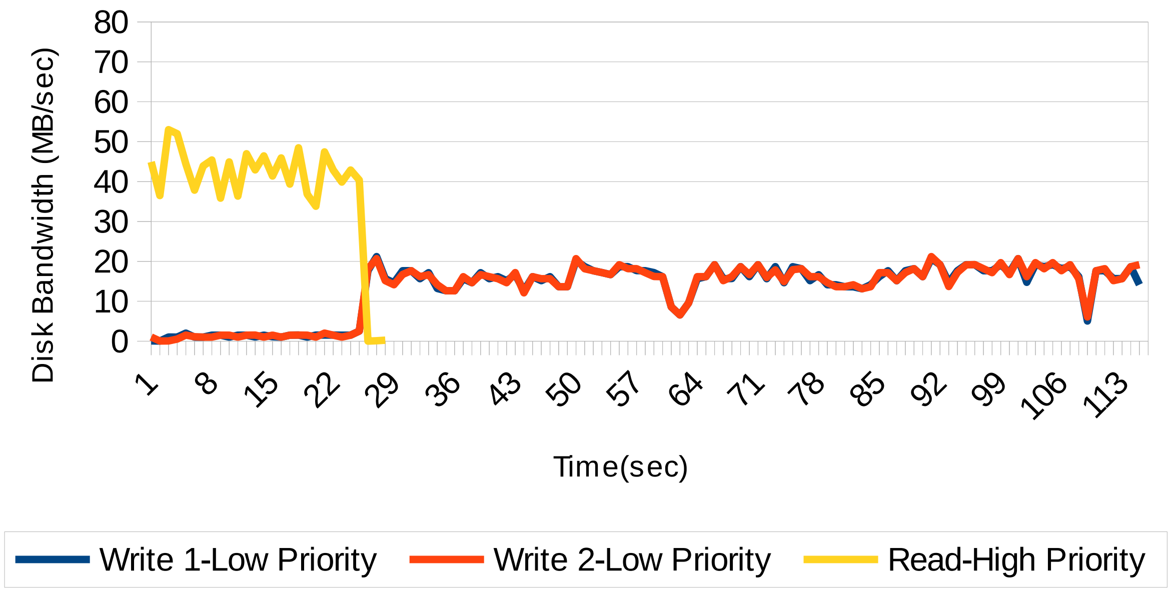
```
Require: $Deadline\ D, TotalDataSize\ R$
Ensure: $Priority\ Pr$
    $LATENCY\ PREDICTOR(R, B)$
 1: for $every\ process\ P_i$ do
 2:     $RemainingData_i = R_i - DataProcessed_i$
 3:     $L_i = RemainingData_i / B_i$
 4: end for
 5: return $Latency\ L$
    $PRIORITY\ MANAGER(L, D)$
 6: $Find\ P_i\ s.t.\ (L_i > (D_i - T_i))\ \&\ D_i\ is\ minimum$
 7: if $(exists\ P_i)$ then
 8:     $Find\ all\ P_{j,(j!=i)}\ s.t.\ (D_j > D_i)\ \&\ (L_j < (D_j - T_j))$
 9:     $Select\ P_j\ s.t.\ (Pr_j > lowest)\ \&((D_j - T_j) - L_j)\ is\ maximum$
10:     if $(exists\ P_j)$ then
11:         $Decrease\ Pr_j$
12:     else                                          ▷ $If\ no\ such\ P_j\ exists$
13:         if $(Pr_i < highest)$ then
14:             $Increase\ Pr_i$
15:         else
16:             $Set\ Pr_i\ to\ lowest$
17:             $Restore\ Pr_j$
18:         end if
19:     end if
20: end if
21: return $Priority\ Pr$
```

Figure 3.3: PriDyn Algorithm

In every iteration of the algorithm, only one out of the three cases that have been discussed is valid. The algorithm considers all non-critical processes P_j and decreases their priority one at a time till the desired performance for a critical process P_i is achieved. The time and space complexity of this algorithm is proportional to N i.e. the number of I/O processes active on the system and it incurs negligible overhead in terms of implementation and execution time at every scheduling interval. As discussed in Section 1.3.2, the degree of multi-tenancy for I/O is limited by the disk bandwidth. Thus, even though a single system may cater to a large number of VMs in typical data-centers, the number of VMs contending for the disk resources cannot be arbitrarily large and N is expected to be a potentially small number. An important feature of our approach is that even though lower priority values are assigned to non-critical processes to get higher disk performance for critical processes, the overall execution time for those processes experiences a small increase in value in comparison with the default case. This



(a)



(b)

Figure 3.4: Disk bandwidth allocation depending upon priorities of processes

is because the critical processes having being assigned higher priority for disk access, get high disk bandwidth and thus finish earlier than the default case. Since the number of processes contending for disk access decreases, the remaining process receive higher disk bandwidth than before and incur less latency. This scenario is illustrated by Figure 3.4b. The latency sensitive read process is depicted by the yellow line and is shown to be using higher disk bandwidth having been assigned higher priority value. The other two write processes represented by the blue and red lines were assigned very low bandwidth while the high priority read operation was running. After the read operation finishes execution, the write operations could get access to higher disk bandwidth which was equally shared among them. This enabled the write processes to finish earlier than the estimated time. Thus, it is observed that even though the *PriDyn* framework provides guaranteed performance to critical applications, it degrades the performance of concurrent applications by merely a small acceptable margin.

3.5 Summary

This chapter presented the detailed design of the *PriDyn* scheduling framework while highlighting its main features and advantages. *PriDyn* considers the latency characteristics of the I/O applications running on a server with the help of the *Latency Predictor* component and ensures that disk resources are appropriately allocated to the applications so as to satisfy the minimum performance requirements. The resources partitioning is achieved with the help of *Priority Manager* component which dynamically adjusts the values as per the feedback from latency predictor. The features of the components has been discussed in this chapter with accompanying figures. The algorithm for the implementation of the scheduler was presented and discussed in detail to explain the functionality. The *PriDyn* scheduler can ensure guaranteed services for I/O applications by efficient management of the disk resources at a fine-grained level. In the next chapter, we present the experimental evaluation of the *PriDyn* scheduler and validation on real world traces.

Chapter 4

Evaluation of PriDyn Scheduler

In the previous chapter, we introduced the design and discussed the implementation details of the *PriDyn* scheduler. In this chapter, we present the performance analysis and experimental validation for *PriDyn* to prove its utility. First, we perform several tests in our experimental setup and present preliminary proof for the functionality of the proposed framework in Section 4.1. However, to analyze the performance of *PriDyn* for actual Cloud storage setups being currently employed for data storage and computations, we use real-world I/O workload traces available from on-line repositories (Section 4.2). Using these traces we perform several experiments which validate that *PriDyn* can achieve better performance for I/O workloads having different functionality by enabling differentiated services on Cloud servers as shown in Section 4.2.3.

4.1 Proof of Concept by Modeling

The functionality of *PriDyn* framework has been validated through extensive experimental analysis. In this section, different use-cases modeling common real world I/O workloads have been employed to evaluate the performance. Typical workloads in a Cloud setup are expected to consist of applications having different functionality and varied service requirements. The applications can be broadly classified on the basis of their latency sensitivity. We have considered two use cases having different number and combinations of latency sensitive as well as other read or write applications such that other Cloud workload combinations could be easily generalized to one of these two scenarios.

4.1.1 Use Case 1: Two write applications, one latency sensitive

We consider a common case where two applications A and B are running on different VMs co-located on the same host. Both the applications are simple write operations with the same data size but different latency requirements. Application A is latency sensitive while application B does not have any strict deadline and is assigned an arbitrarily large value of deadline. When these applications are run on their respective VMs simultaneously, by default they are assigned *best-effort* priority for the disk on the host. Table 4.1a shows the latency values for the default case. Also shown are the latency values for the best case scenario i.e. when the latency sensitive application is given highest priority and the other application is given lowest priority. This is the lower bound of latency value for any application that is running on the system with another application contending for the disk at the same time. With this information, we run the two applications on the VMs, setting different values of deadline for application A in decreasing order in successive experiments. Figure 4.1a shows the latency values achieved for the processes. The latencies for the latency-sensitive application A are depicted in yellow and the other application B is shown by the green line. The x-axis represents the successive experiments performed with decreasing values for the deadline desired for application A as denoted by the blue line. The average value of latency for application A is also depicted in the figure by the red line for the purpose of comparison. The corresponding values for latency obtained for the applications by the *PriDyn* algorithm for different experiments are shown. To understand the results, we can consider three distinct cases here.

In the first case, the value of deadline specified for A is greater than the latency value that it achieves in the default case. Here, the deadline will be easily met with the default priorities as shown in experiments 1-3. The second case is when the deadline value specified for A is less than the latency value achieved in default case. Experiments 4-6 in the figure represent this case. Here, it is clearly seen that *PriDyn* reduces the latency value of application A such that it is less than the deadline value. This is achieved by assigning lower priority value to application B (*Case 1* of algorithm). In the third case, the deadline value specified for A is lower than the best-case latency value achievable on the system (experiments 7-9). In this case, it is not possible to meet the desired deadline for application A on this system. The algorithm assigns it the lowest priority value for disk access (*Case 3* of algorithm) and the latency for application exceeds the specified deadline. As observed in Figure 4.1a, the framework is able to meet the desired deadlines for the latency sensitive application for all values within the performance bounds of the system.

4.1.2 Use Case 2: One latency sensitive read and two write applications

This scenario models a typical use case in a Cloud setup where we consider one read intensive, time bound, video streaming application and two other bulk write applications (such as scientific

Table 4.1: Latency Values

Latency (Time in seconds)	Application A	Application B
Default case	88.76	88.10
Best case	55.07	89.08

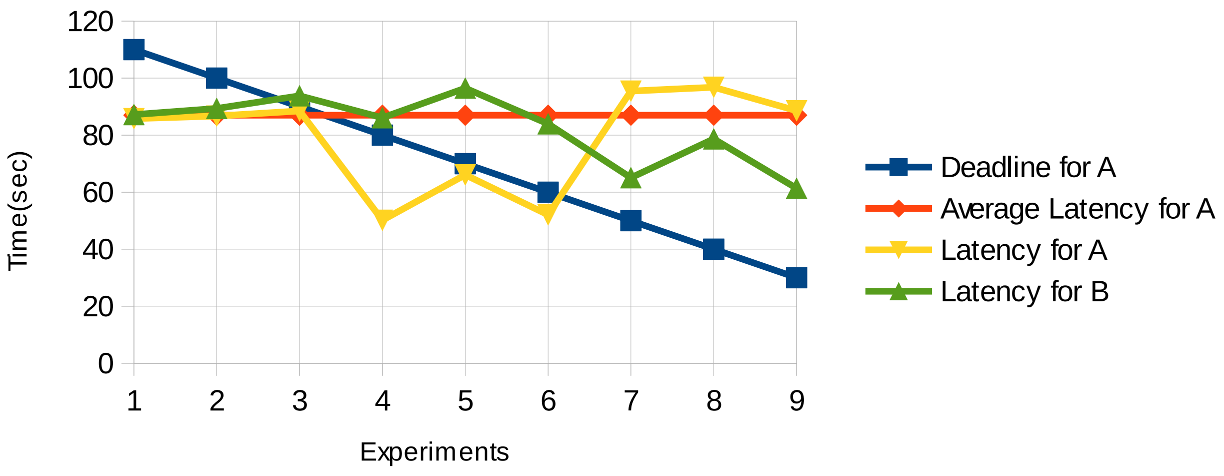
(a) Use Case 1

Latency (Time in seconds)	Read	Write 1	Write 2
Default case	155.97	133.94	138.73
Best case	57.95	150.34	147.93

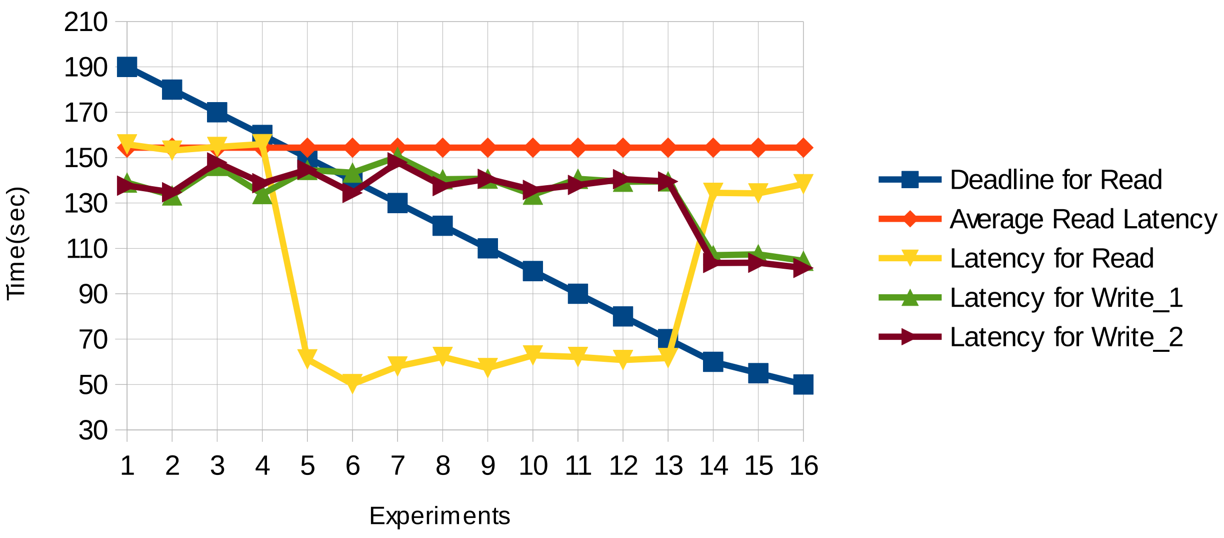
(b) Use Case 2

applications) having long execution times. Table 4.1b shows the latency values achieved for the applications in default and best case scenarios (with respect to the read application). Figure 4.1b shows the latency values achieved by the applications for the experiments. The deadlines for write applications were assigned arbitrary high values and the deadline for read application was varied from a higher to lower value as shown by the blue line in the figure. The latency values obtained for the latency-sensitive read application are depicted in yellow while the average latency is represented by the red line. Experiments 1-4 depict the case where the deadline specified for the read application is more than the latency achieved in default case and thus deadline is easily met. Experiments 5-13 represent the case where the deadline value is lower than the default latency but higher than the minimum achievable value according to system bounds. As can be seen from figure, the latency for read application is lowered by the algorithm and deadline is satisfied in these cases. In experiments 14-16, the specified deadline values are lower than the minimum achievable value on the system. The read application is assigned lowest priority in this case and its deadline is violated. It can be observed from Figure 4.1b that a wide range of deadline values is being satisfied for the read application with the help of *PriDyn* algorithm, which was not possible in the default case.

These experimental results show that *PriDyn* framework efficiently manages allocation of disk I/O resources in such a way as to provide differentiated services to various applications according to their performance requirements. In our experimental setup, we realized a decrease of 35% in latency value on an average for the write application in Use Case 1 and 62% average decrease in latency of the read application in Use Case 2. The performance of *PriDyn* has been illustrated with the help of two generic scenarios, other variations with more number of VMs and different kinds of application requirements can be modeled as one of the three cases of our algorithm and are expected to show similar results. We perform experiments on some of the



(a) Use Case 1



(b) Use Case 2

Figure 4.1: PriDyn performance on benchmarks

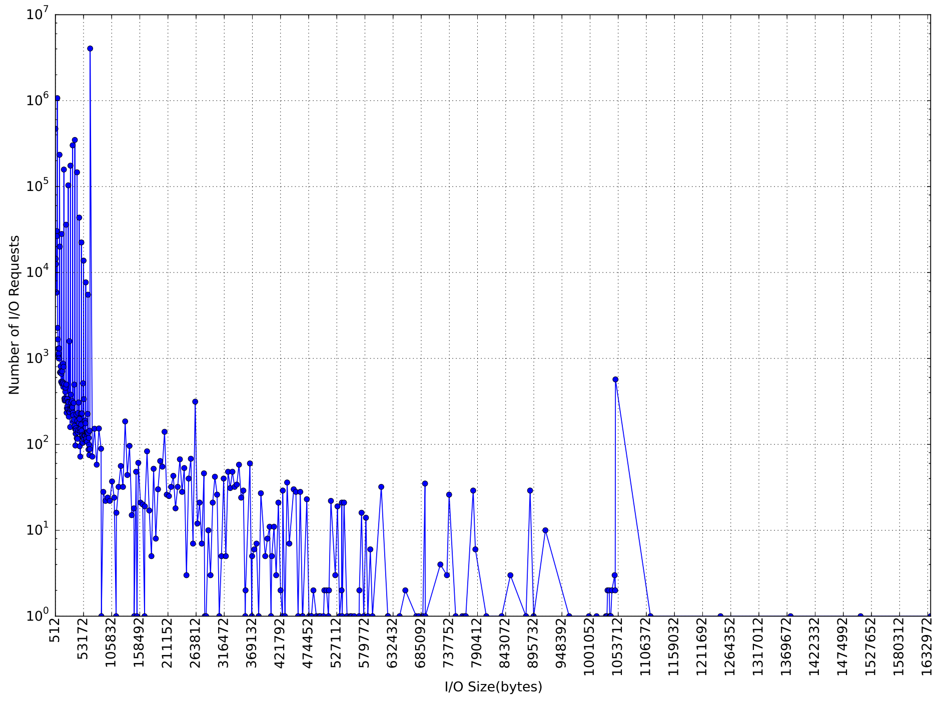
possible different combinations of actual real world I/O traces in the next section.

4.2 Validation with Real I/O Traces

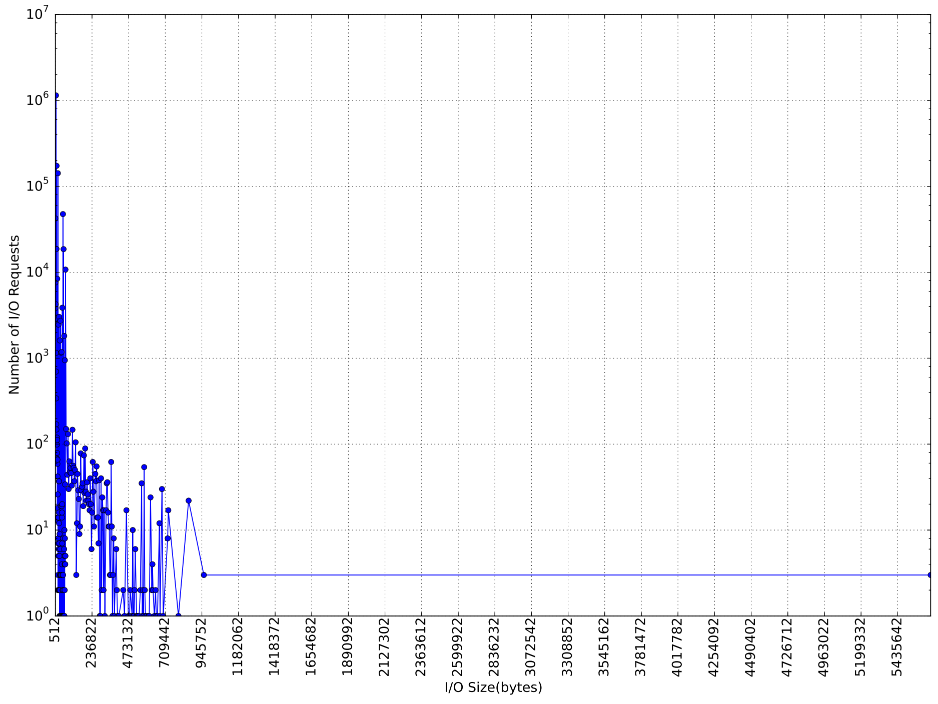
In previous section, the functionality of *PriDyn* Framework was demonstrated with the help of two use cases modeling real life applications. In order to prove the practical utility of the proposed framework for actual Cloud setups, it is important to validate its performance on real I/O workloads. To enable this, actual I/O traces were considered to test the performance for different types of I/O workloads. We have modeled the I/O requests based on traces available at SNIA IOTTA Repository [70]. These are block I/O traces from the servers at Microsoft Cambridge [71] handling various enterprise workloads like web service, email, file service etc. and thus these can be considered as a good representation of the varied I/O workloads in a typical Cloud data-center. The information about the timestamps and sizes of I/O requests available in these traces was used to replay the trace requests with the help of *IOzone* benchmark on the experimental setup.

For the experiments in this section, the I/O traces from 2 servers having different functionality have been chosen - Web server used for handling common web applications and Research server that is used to serve batch operations typical of research workloads. The web server traces have strict deadlines for their requests as they are latency sensitive whereas the research server I/O requests usually have larger deadlines and can tolerate some deviation from the assigned deadline values. The trace data was analyzed to derive the characteristics of the I/O workloads. Figure 4.2a and 4.2b show the frequency distribution for the size of I/O requests for the web and research traces under consideration. It can be seen that there is wide variation in the request size frequency for the web traces since the workloads they handle are very dynamic in nature. On the other hand, for the research server traces, most of the requests are small in size except for a few cases of outliers. Figure 4.3a and 4.3b show the variation of I/O request size with time for the web server and research server traces respectively.

For performing the experiments we have considered the following scenario - Multiple I/O applications with diverse requirements (as represented by the different traces) execute simultaneously on different VMs on the same physical server. Each application is modeled as a stream of I/O requests contained in the corresponding trace file, which are consecutively scheduled on the VM on a continuous basis. As soon as any I/O request finishes on any of the VMs, the next request is read from the trace file corresponding to that VM and it is scheduled on the system. Since the size of I/O requests is very small and the response time is too short to be noticeable on our setup, the request I/O sizes have been scaled up by a factor of 100 so as to enable correct measurements for response times during experiments. Since the traces did not

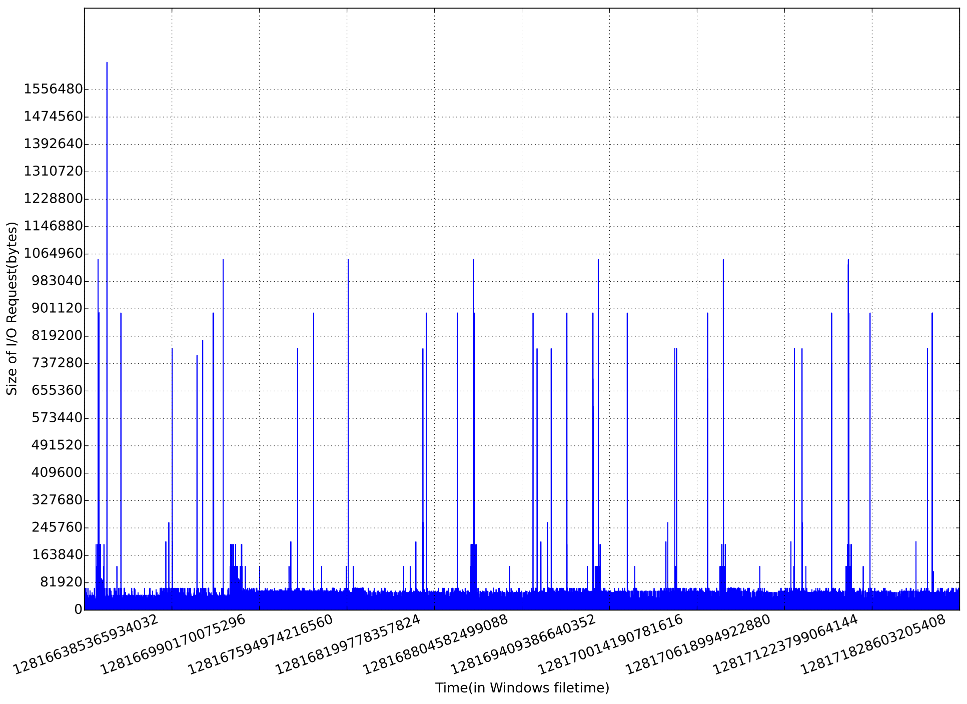


(a) Web Traces

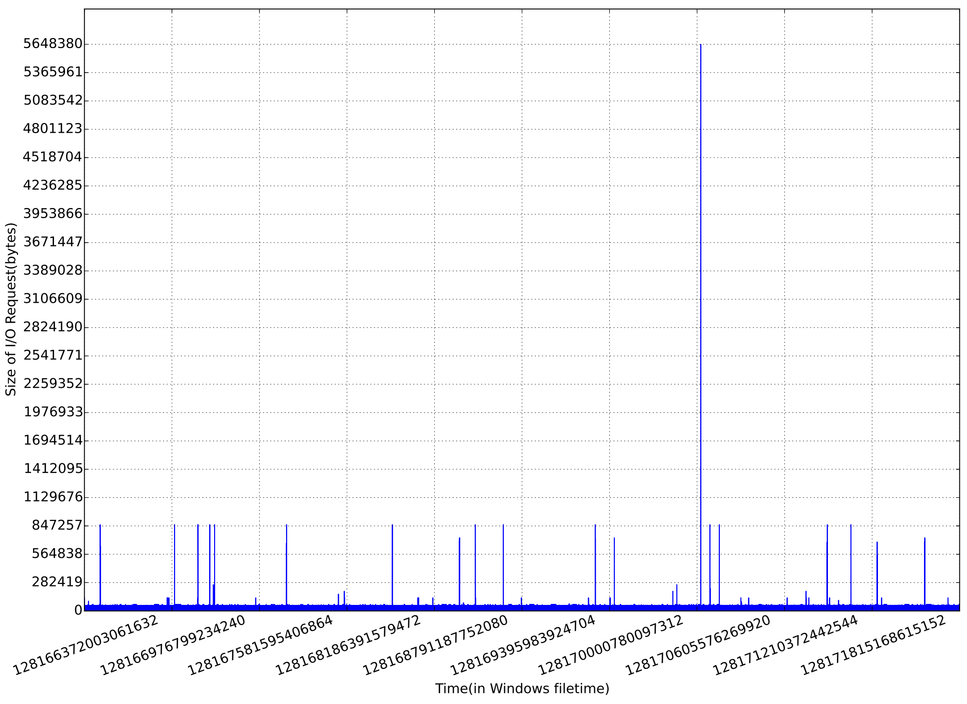


(b) Research Traces

Figure 4.2: Frequency distribution of size of I/O requests



(a) Web Traces



(b) Research Traces

Figure 4.3: Time distribution of size of I/O requests

contain any information regarding the deadlines, all the I/O requests are assigned deadline values according to a deadline assignment scheme as described in Section 4.2.1. The simultaneous execution of multiple I/O streams on different VMs is achieved remotely by a scheduling script running continuously on the server machine. This enables modeling of real world scenarios where several users may concurrently run applications on VMs oblivious to the presence of other co-located tenants. The response time for each request is logged in a separate file for each virtual machine along with the start time of the request and the assigned deadline values in our setup. This log file is used to then calculate the percentage of missed deadlines for each of the I/O stream corresponding to the VMs. Details about the configuration and features of the setup for running real I/O traces including the techniques for deadline assignment and avoiding starvation of I/O requests are discussed further in Section 4.2.1 and 4.2.2. The results obtained from executing the *PriDyn* framework on real I/O traces are described in Section 4.2.3 with discussions on the performance improvements achieved.

4.2.1 Deadline Assignment

In order to assign deadlines to the I/O requests obtained from the traces, we have employed the techniques commonly used in literature [72, 73, 57, 74, 75]. For every request, the minimum time required to finish the I/O request on the system, if it was assigned the full disk bandwidth is denoted by *Makespan* and this is used for calculating the deadline. In case of a multi-tenant system such as the experimental setup under consideration, the bandwidth that will be allocated to an application will depend on the number of applications co-located with it i.e. the degree of multi-tenancy. As we have shown in Section 1.3.2 (Figure 1.3), in a multi-tenant environment there is loss of disk bandwidth due to contention for resources and this bandwidth loss is represented by a parameter $BWloss$. The parameter $BWloss$ is proportional to the number of co-tenant applications (VMs) denoted by N such that the value of $BWloss$ increases as N increases. This can be seen from Figure 4.4, where the stacked bars denote the average disk bandwidth values for the VMs co-located on the same host for varying number of VMs and the total bandwidth achieved in each case is denoted by the cross marks. For a given value of N, the value of $BWloss$ is obtained as the loss in the total bandwidth value as compared to the case of a single VM i.e $N = 1$. Based on these parameters, the expected completion time or *Makespan* of any I/O request on the system will be calculated as follows:

$$Makespan = IOSize/(MaxBW - BWloss)/N \tag{4.1}$$

where *MaxBW* is the maximum disk bandwidth that can be allocated to a VM on the system, $IOSize$ is the data size of the I/O request under consideration and $BWloss$ is determined by the number of co-tenant VMs.

Now, different I/O requests may have different requirements of time sensitivity or urgency depending upon their characteristics. This feature is assumed to be specified by the users requesting for disk resource for their I/O applications. We can model these user requirements in terms of the delay tolerance of the application by another parameter δ while assigning deadlines as :

$$Deadline = Makespan + (Makespan \times \delta) \tag{4.2}$$

Equation 4.2 assigns deadlines to the requests as per their latency sensitivity characteristics such that requests with higher delay tolerance are assigned larger deadline values. In this way our deadline assignment scheme is taking into account both system constraints as well as application features while assigning deadlines to I/O requests. The system constraints and overheads due to multi-tenancy are being considered by the $BWloss$ parameter while the delay tolerance of the I/O request as per the application characteristics is being taken care of by the δ parameter.

To decide appropriate values of δ for a given setup, the effect of different δ values on the performance of the scheduled applications (i.e. constant value of $BWloss$) was explored through experiments. For a set of δ values, measurements were made for the number of application requests whose deadlines were not met. Figure 4.5 shows the percentage of deadlines that were violated for different values of δ when two streaming applications represented by web server traces are scheduled to run concurrently on the system. Here, the value of N is 2 and $BWloss$ is set to 17 (corresponding to loss in bandwidth from 54 MB/s in case of 1 VM to 37 MB/s in case of 2 VMs). It can be seen clearly from the figure that if the value of δ is small, the deadlines are stricter and therefore there are more deadline violations. On the other hand, if the δ value is large, then the deadlines are lenient and the percentage of deadline violations has a lower value. According to the figure, for the value of δ as 0.2, all the deadlines are being violated and for δ as 1.0, all the deadlines are comfortably satisfied. The experiments in this analysis were therefore conducted within this range of δ with increments of 0.1 in value.

4.2.2 Avoiding Starvation of Large I/O Requests

According to our deadline assignment scheme, the deadlines for requests are approximately proportional to their I/O sizes, which is appropriate since larger I/O requests will usually take longer time to complete. However, in some cases this can lead to starvation of large I/O requests in the current setup. Given a set of requests running on the system, the *PriDyn* algorithm will

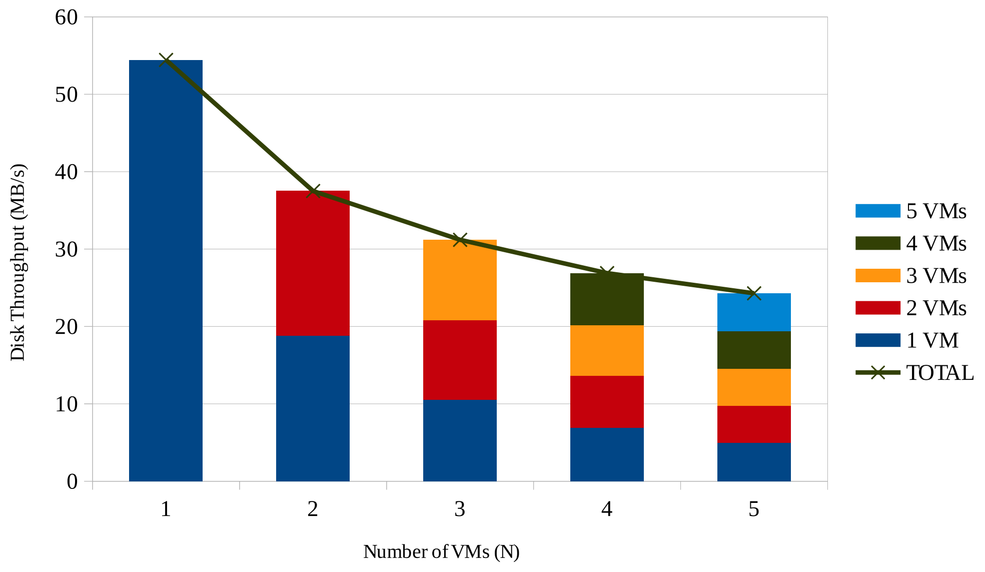


Figure 4.4: Calculation of $BWloss$ parameter for different N

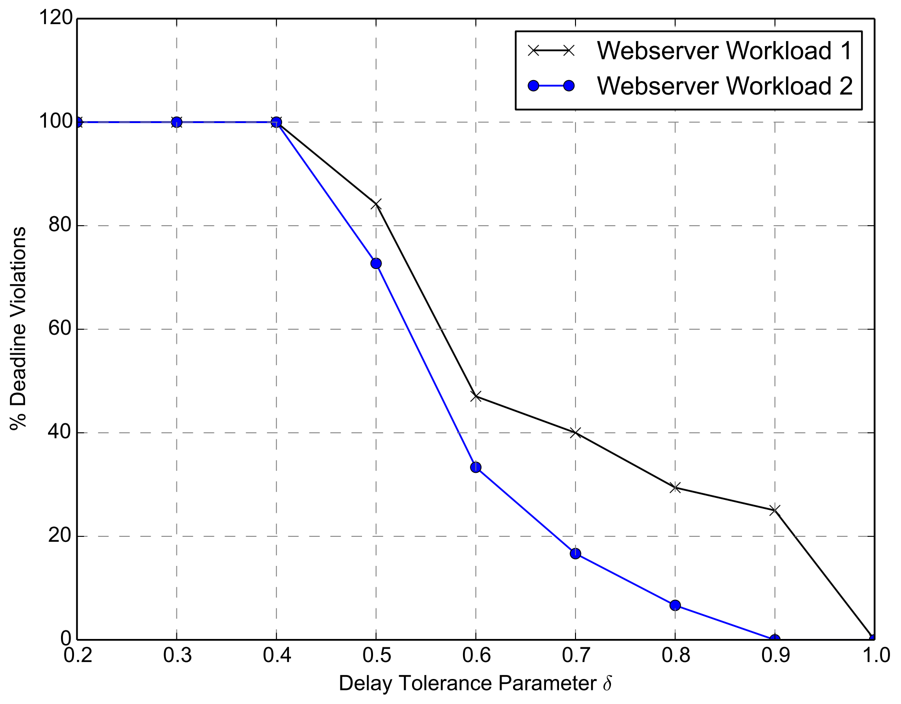


Figure 4.5: Variation in deadline violations with delay tolerance parameter δ

give higher priority to the request whose deadline is closer, such that the request does not miss its deadline value. In such a scenario, if there is a large I/O request having higher deadline value, it will be given lower priority as compared to a smaller I/O request running concurrently with it. With multiple applications running in parallel if there arises such a case where one of the I/O streams has larger I/O request sizes while the others have small I/O request sizes, then it is possible that the larger I/O request is not given high priority for a long time while the smaller requests are being swiftly handled and executed on the system by the scheduling algorithm. This may lead to starvation of the requests of the applications having larger sizes. In order to avoid such an undesirable situation, the scheduling algorithm for *PriDyn* has been enhanced to include a time check for possible application starvation. If an I/O request does not complete execution by the time of its assigned deadline value (as measured from the moment when it is scheduled on the system), then it is temporarily given the highest priority such that it gets higher disk bandwidth and completes with minimum possible delay from the time of its original deadline value. This approach successfully prevents the starvation of larger requests and also ensures that even though large requests may miss their deadline value, but the value of lag between the deadline and the actual response time is minimized.

4.2.3 Results

In order to illustrate *PriDyn* functionality, we consider a scenario where the execution of a web server workload having strict deadlines was modeled on a system along with a research workload which is assumed to have no deadlines for its requests. The selection of such a workload combination on a physical server is essential in order to leverage the difference in latency sensitivity of I/O requests to ensure performance and this is assumed to be taken care of by an intelligent admission control mechanism in the Cloud data-center. The I/O requests were assigned deadline values as per the deadline assignment scheme described earlier. The research I/O requests were assigned high value of delay tolerance parameter δ since they are not latency sensitive whereas web server requests were assigned smaller δ value (0.3 here). Since the I/O request streams were executed concurrently on the same server, contention for the disk bandwidth caused performance degradation for both the request streams. Figure 4.6 shows the deadline values and the response times for the I/O requests of the latency sensitive application i.e. the web server requests executing on the system both without and with the *PriDyn* algorithm. The blue bars denote the deadline values while the red and green bars show the response times that were obtained for the I/O request without and with the use of *PriDyn* scheduling respectively. It is clearly seen that the response times have lower values with *PriDyn* algorithm since it performs dynamic priority adjustments for the requests based

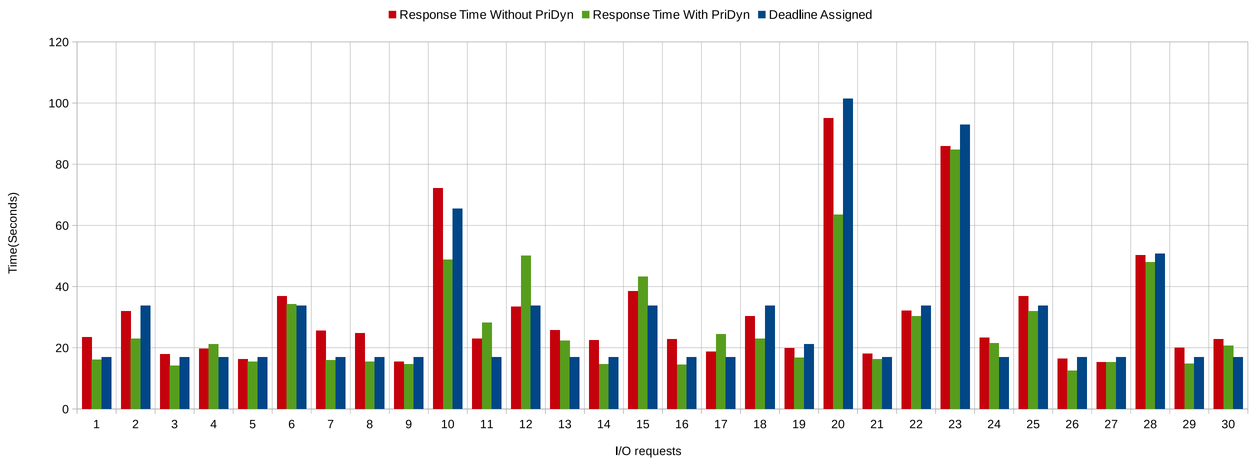
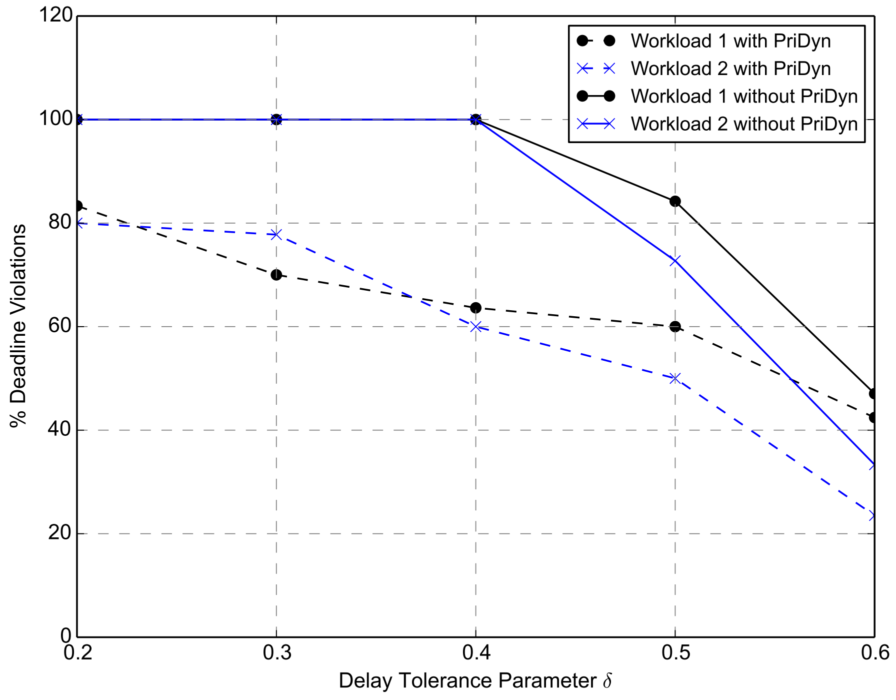


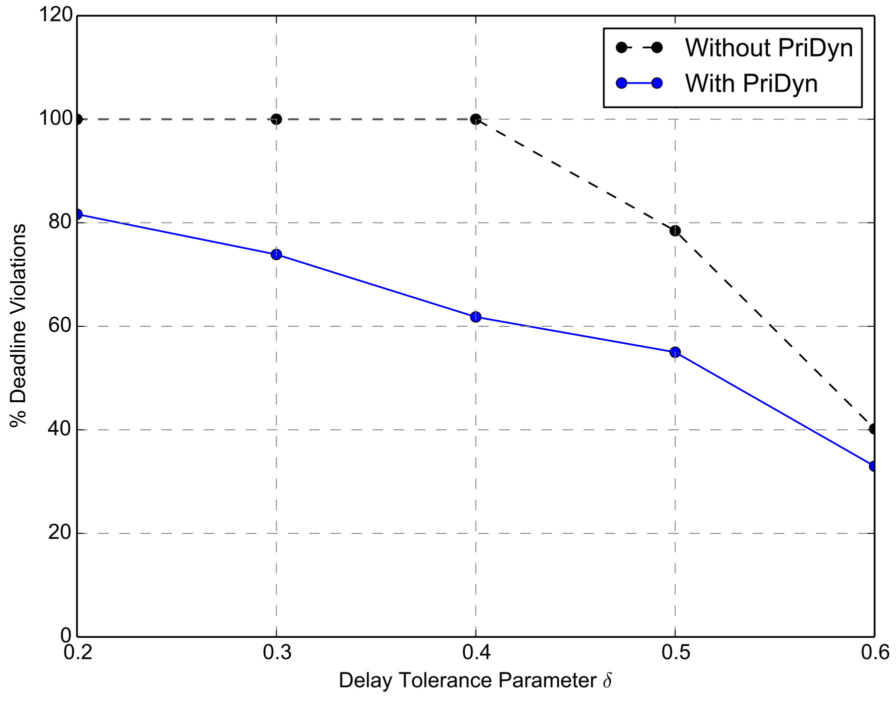
Figure 4.6: Improvement in response time for latency sensitive application

on deadlines. At every scheduling interval, the *PriDyn* algorithm considers the set of I/O requests being executed at that instant, calculates their expected latency values based on their disk throughput and adjusts the priorities of the I/O processes depending upon the deadline values that have been assigned to the requests. As soon as a new request is started, the new set of requests is again considered for disk bandwidth allocation. In the case under consideration, higher disk priority values were assigned to the web server requests (having strict deadlines) in order to achieve the desired service requirements. The number of deadline violations was reduced from 85% to 35% for web server requests with the help of *PriDyn* framework.

In another scenario, two web server I/O workloads represented by different web server traces were modeled to execute simultaneously on the same setup. The delay tolerance parameter δ was set to the same value (0.3) for both the I/O streams for simplicity since the workloads have similar characteristics, although it could be varied for representing different degrees of latency sensitivity based on the application requirements. The *PriDyn* algorithm was executed continuously along with the I/O requests on the server to assign the disk bandwidth to the two contending application workloads based on the deadlines of the set of requests under consideration at a given scheduling instance. Dynamically readjusting and balancing the allocation of disk bandwidth by *PriDyn* enabled better utilization of disk resources leading to an overall improvement of performance for the concurrent I/O applications even when both of them were latency sensitive. This can be seen from the Figure 4.7a where the percentage of deadline violations for workload 1 are shown with black lines and those for workload 2 are depicted by the blue lines. The deadline violations have been reduced for both the I/O workloads running on different VMs with the help of *PriDyn* (shown by the dotted lines) as compared to the



(a) Performance improvement for applications



(b) Overall performance improvement for system

Figure 4.7: PriDyn Performance on I/O traces

case where the requests were executed without *PriDyn* framework. Similarly, the percentage of deadline violations for the overall system is brought down considerably by scheduling the requests with *PriDyn* as shown in Figure 4.7b. The experiments were conducted for a number of δ values and lower deadline violations were observed for all the values of δ in range. We note that *PriDyn* can lower the number of deadline violations considerably even for the δ values where all the deadlines were being missed earlier. This shows that better performance can be obtained for the workloads even in the presence of strict deadlines through assignment of dynamic disk priorities by our framework.

We conclude this section by observing that the *PriDyn* framework has been broadly validated for a wide spectrum of I/O workload combinations using both I/O benchmarks as well as modeling of real I/O workload traces. It has been shown to be performing consistently well for all scenarios considered thus proving its efficacy for practical Cloud environments.

4.3 Summary

This chapter presented the analysis and experimental evaluation of the proposed *PriDyn* algorithm. The framework was validated by modeling on simple test cases to show the proof of concept. Real-world I/O workload traces were also used to illustrate the feasibility and performance benefits derived from the framework for practical Cloud storage setups. Results from the experiments performed with real workload traces show that *PriDyn* achieves substantial decrease in the percentage of deadline violations for concurrent workloads, proving its utility for real world environments. However, *PriDyn* has made the assumption that the workloads executing on the same host differ in their latency characteristics. *PriDyn* could then leverage this disparity in service requirements to promise QoS guarantees for all the applications running concurrently through dynamic allocation of disk resources. For ensuring a proper combination of workload on the host, it is essential to have a meta-scheduling framework for deciding the placement of applications at the data-center level. In the next chapter, we will illustrate the need for extending the *PriDyn* scheduling for data-center level and propose the design of *PCOS* framework that performs meta-scheduling for the new application requests in Cloud systems.

Chapter 5

PCOS Framework

Consolidation of I/O workloads on physical servers in virtualized environments needs due consideration of the impact of virtualization on the performance of the applications. It is clear from the observations made in the previous chapters, that merely matching the I/O capacity of a server to the requirements of the applications does not necessarily ensure fulfillment of performance guarantees. This is attributed to the overheads of virtualization on consolidated servers and also to the fact that most I/O workloads have different characteristics and exhibit variability in their resource requirements during a transaction session.

As described in the previous chapter, *PriDyn* leverages on this aspect of I/O workload behavior to provide differentiated services to a given set of applications executing concurrently on a server. However, *PriDyn* can only attempt to provide performance guarantees for applications that were already scheduled on the system. In the worst case, if the deadline of an application cannot be satisfied by any possible rearrangement of disk priorities due to system resource constraints, then it is recommended to migrate that application to a different server. This reactive approach is not preferable for large scale data-centers due to the high overheads associated with the migration of VMs, especially for I/O applications [76]. It is essential that a proactive approach for scheduling I/O applications is adopted wherein the combination of applications scheduled on a system should be chosen intelligently by the data-center meta-scheduler. Experimental results have indicated that scheduling the correct mix of workloads can ensure good resource utilization without faulting on the performance guarantees of the consolidated applications. Hence, before placing the VM on to a host, it is important to assess if the host can provide the desired SLAs dictated by all the VMs destined for it. A new application should be accepted for execution on a host only after ensuring that performance of the system will not be compromised due to the new application.

Cloud data-center policies in current practice consider new application for scheduling in terms of their average requirements over the entire execution period. For a new I/O request being considered for scheduling on a server catering to already running applications, the meta-scheduler should not only match the available resources with the average requirement but also ensure that the peak workload requirements for the new application does not cause interference and performance degradation for the previously scheduled applications. Therefore, a Cloud scheduler needs to adopt a multi-dimensional scheduling approach for the applications where the performance aspects of the previously running applications should be simultaneously considered along with the requirements of the new applications. In addition, the Cloud scheduler should be able to obtain good workload consolidation on the servers for ensuring optimal utilization of the resources and achieve energy efficiency for the data-center.

In this chapter, we introduce a novel admission control and scheduling framework named ***P****rescient* ***C****loud I/****O*** ***S****cheduler* (*PCOS*) for obtaining better mix of workloads to achieve consolidation for Cloud systems. *PCOS* aims to maximize the utilization of disk resources without adversely affecting the performance of the applications scheduled on the systems. This scheduler follows a conservative methodology in which migrations of I/O applications among the systems is strictly avoided. By anticipating the performance of the systems running multiple I/O applications, *PCOS* can prevent the scheduling of undesirable workloads on them. The *PCOS* framework includes the *PriDyn* scheduler as an important component and utilizes the dynamic disk resource allocation capabilities of *PriDyn* for meeting its goals.

Section 5.1 discusses the performance of different combinations of I/O workloads to motivate the need for scheduling applications in an optimal manner. The *PCOS* framework has been introduced in Section 5.2 and its design and functionality is discussed in detail in Section 5.2.1. Section 5.3 discusses the algorithms for implementation of the framework in detail to understand the features better. We also analyze the functionality of the framework by demonstrating how *PCOS* selects the workload mix for a server such that the overall resource utilization is maximized without any compromise on the application performance. When this scheduling policy is replicated across the data-center, overall consolidation for the servers can be achieved as discussed in Section 5.4.

5.1 Characteristics of I/O Traces

In this section, we analyze the performance of a system under different kinds of I/O workloads and emphasize the need for a meta-scheduling framework that achieves proper placement of multiple I/O applications on a single server. We perform experiments using the real world I/O traces as described in the previous chapter. Figure 5.1a and 5.1b show the characteristics of the I/O traces considered in terms of the frequency distribution for the size of I/O requests

Table 5.1: I/O Workload Combinations

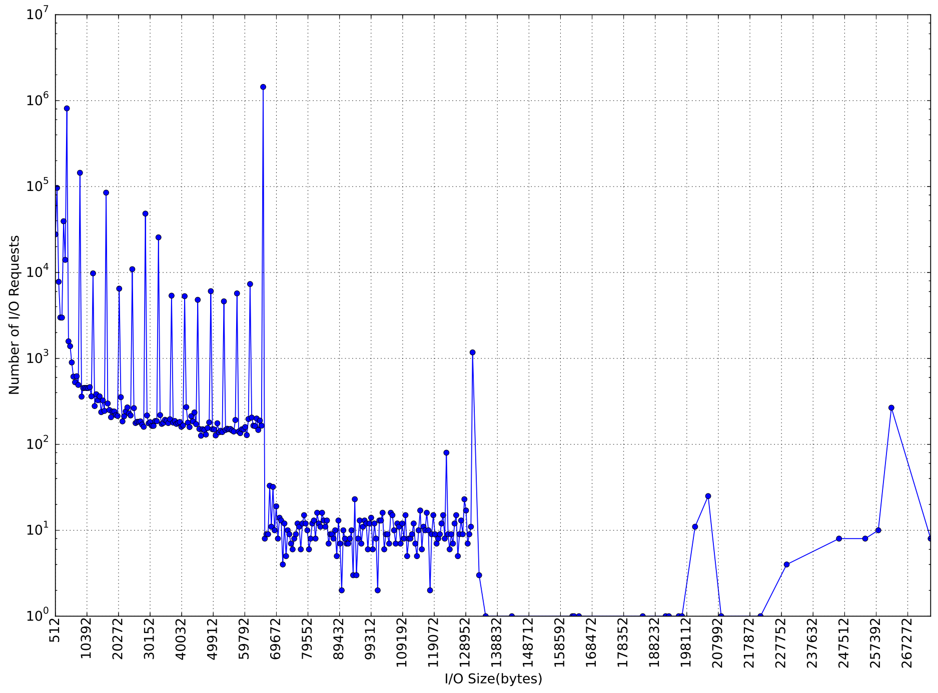
	Features	*Application A*	*Application B*	*Application C*
Case 1	Latency sensitive	Yes	Yes	Yes
	Disk Priority	Default	Default	Default
Case 2	Latency sensitive	Yes	Yes	No
	Disk Priority	Default	Default	Low

for media and research traces. It can be seen that there is wide variation in the request size frequency for the media traces since they the application is very dynamic in nature. On the other hand, for the research server traces, most of the requests have smaller size except for a very small number of outliers. Based on the characteristics of the applications, the media server traces were assigned strict deadlines for their requests whereas the research server I/O requests were assigned large deadline values since they can handle delays in completion time. Since the research server requests are small in size and can tolerate deviations from the deadlines, they seem to be appropriate for scheduling along with applications such as media streaming since the resource requirements for these applications may complement each other to enable server consolidation by better utilization of disk bandwidth. This is illustrated with the help of experiments in the next section.

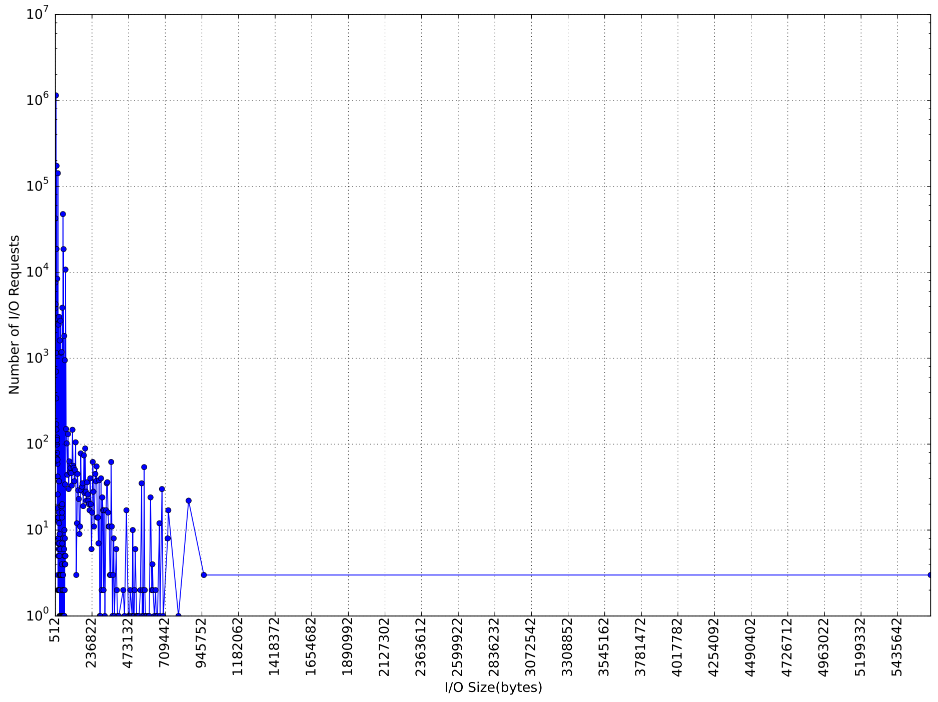
5.1.1 Need for I/O Meta-scheduling

In order to illustrate the importance of ensuring an optimal mix of I/O workloads while deciding the placement of applications on the servers, we demonstrate the performance characteristics of different combinations of I/O applications on our setup with the help of various experiments on real world traces. First, two latency sensitive I/O applications (represented by media server traces) were executed simultaneously on the server such that they were able to achieve the desired performance as per their latency requirements i.e. most of the I/O requests were able to meet their deadlines. With the two I/O applications contending for disk bandwidth, an additional application was scheduled on the system. Workload combinations having different features as specified in Table 5.1 were analyzed.

In Case 1, a third I/O application (application C), also being latency sensitive, was scheduled on the system to execute concurrently along with the two previously running applications (A and B). The response time for the I/O requests of applications A and B was measured before and after the scheduling of the new application. Figure 5.2a shows the pattern of response times for application A for its I/O requests. The x-axis represents the I/O requests and the corresponding response times are indicated with the blue line. Also, the deadlines have been shown by a

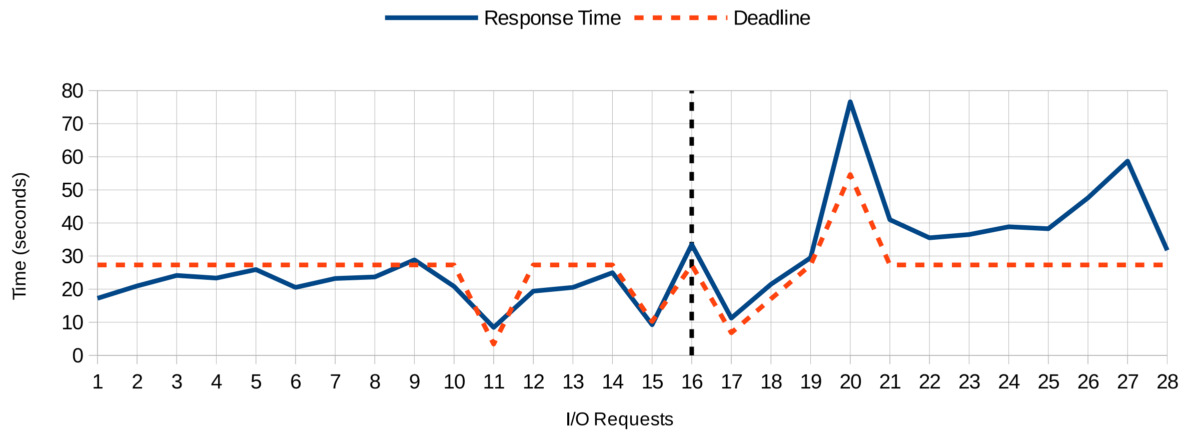


(a) Media Traces

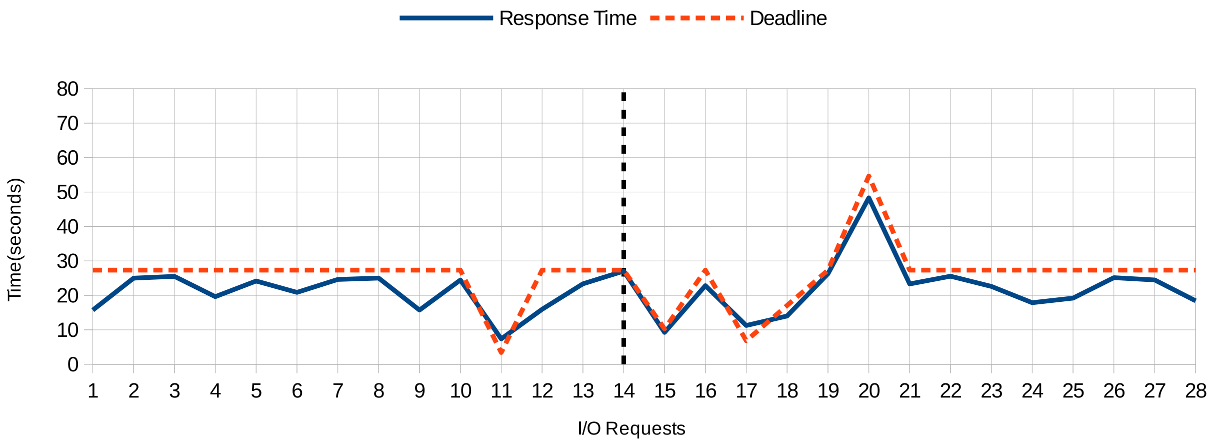


(b) Research Traces

Figure 5.1: Frequency distribution of size of I/O requests



(a) Case 1



(b) Case 2

Figure 5.2: Response Times for Application A

dotted red line. The time of scheduling of application C is indicated in the figure with a dotted vertical line. It can be seen that before scheduling Application C, when only application A and application B were running on the system, the response times for I/O requests of application A were below the deadline values. However, when application C started execution, the response time started increasing to higher values thereby violating the deadlines. Application B also showed similar results although they are not shown here to avoid repetition. The number of deadline violations was very high for application C as well. Such application behavior was observed since all the applications in Case 1 had strict latency requirements and required high priority disk access at the same time thereby leading to performance degradation for all of them.

In Case 2, the third I/O application was chosen such that it was not latency sensitive (represented by research server trace) and could be executed with low priority to disk access without affecting its performance. It was observed that in such a situation, the response time of the previously running applications was not affected to a great extent. Although some deadline violations were observed, but the overall percentage was much less as compared to Case 1. This can be seen from Figure 5.2b where the response times of the requests of application A remain below the deadline values both before and after the start of execution of application C. Similar pattern was observed for application B as well. Since application C did not have strict deadlines, it also achieved good overall performance in this setup.

For further illustrating the effect of I/O workload scheduling in case of latency sensitive applications, comparison between the values of response times for application A for both Case 1 and Case 2 is shown in Figure 5.3. It is clearly seen that the deadlines were largely violated for the I/O requests in Case 1 (response times shown in blue are larger than the deadline values) when all the latency sensitive applications were running at the same time. On the other hand, in Case 2, majority of the deadlines were satisfied with the response times (shown in green) being smaller than the deadlines since one of the applications was delay tolerant.

Finally, the overall comparison of performance of applications in the two cases in terms of the percentage of deadline violations is shown in Figure 5.4. The bars show the percentage of deadline violations corresponding to the applications for both Case 1 and Case 2. It is evident that the deadline violations decrease considerably in Case 2 for all the applications as compared to Case 1 (80% to 10% for application A). Also, in Case 2 no deadline violations occur for the third application since it is not latency sensitive whereas it incurred around 40% deadline misses in Case 1 when it had strict deadlines like the other two applications. Thus, it is evident from the observations made with the experiments in this section that the combination of I/O applications being executed on a system affects their performance to a great extent. An intelligently chosen I/O workload combination can not only assist in obtaining the desired performance but also attain a high degree of multi-tenancy through better utilization of disk resources. In the next few sections, we describe the design and implementation of the proposed framework *PCOS* which strives to achieve the above stated goals.

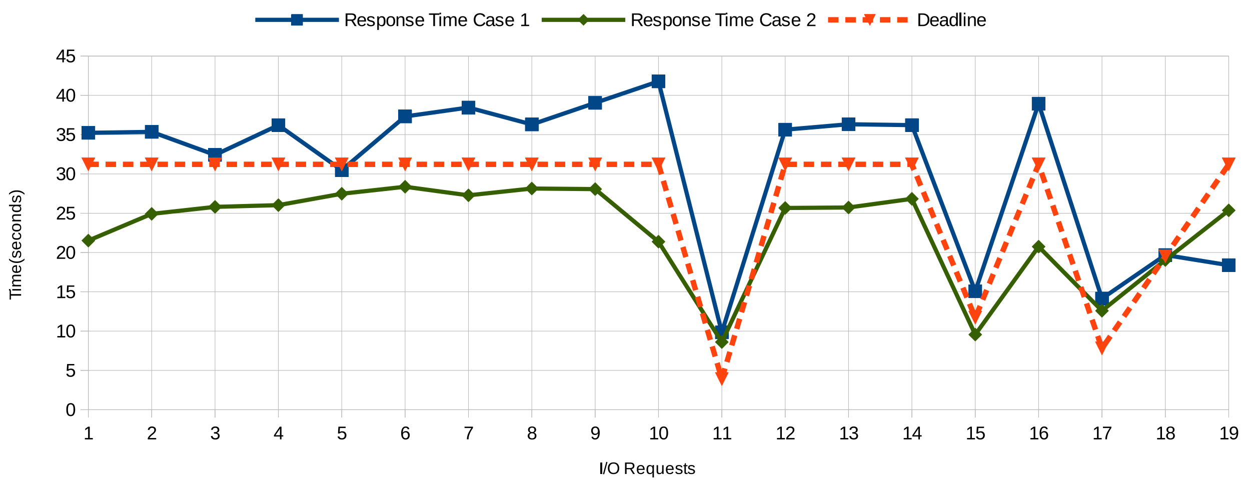


Figure 5.3: Performance of Application A for Case 1 and Case 2

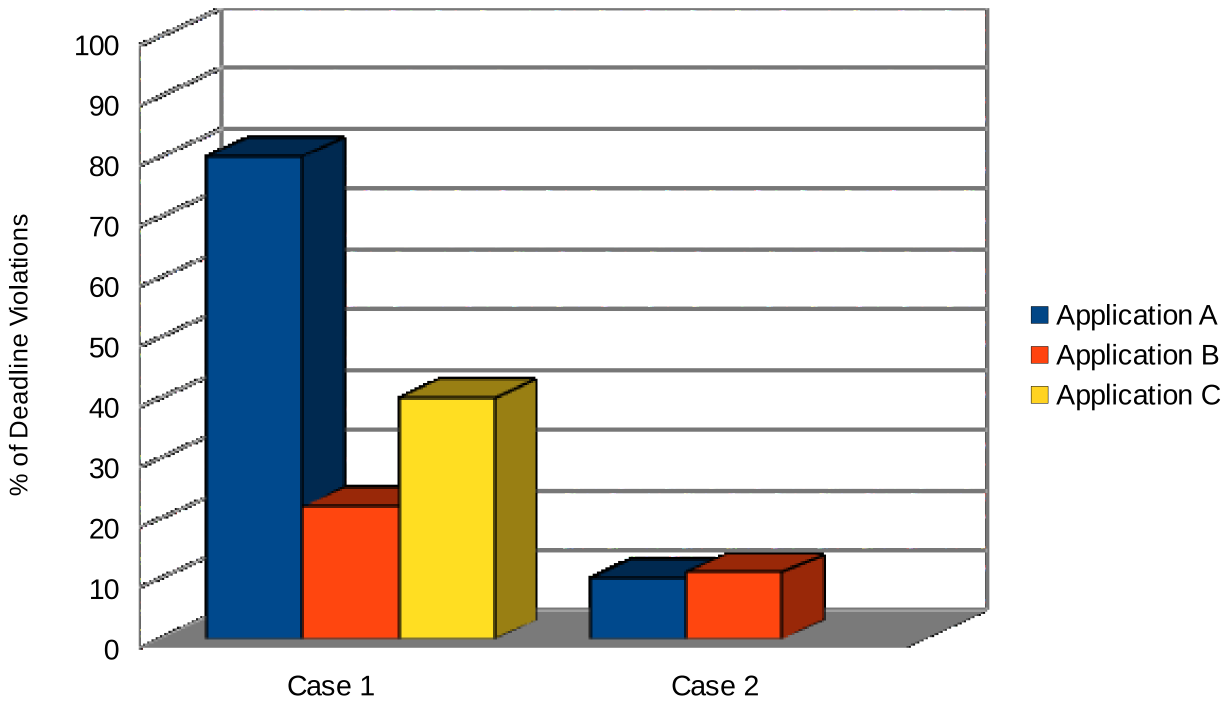


Figure 5.4: Deadline Violations for Applications

5.2 Prescient Cloud I/O Scheduler

The details of the proposed framework are discussed in this section. Figure 5.5 shows the overall design and placement of *Prescient Cloud I/O Scheduler* at the data-center level. There are multiple physical servers with local storage disks and variable number of VMs executing on them concurrently. I/O applications running in the VMs on each physical server are locally managed by a hypervisor or a virtual machine monitor (VMM) in a typical Cloud environment. The hypervisor is responsible for allocating the physical resources to the VMs and managing their execution. In this setup, scheduling of the I/O applications on the servers is handled by the *PCOS* framework. *PCOS* can be integrated with the global resource manager of the data-center that centrally manages the allocation of physical resources to the applications. For every new I/O request, the *PCOS* framework searches for a suitable server where the request can be scheduled without performance degradation. There are two main components of *PCOS*- *Admission Controller* (*AdCon*) and *PriDyn* disk scheduler which function in a collaborative manner. They are assumed to be continuously executing on the servers having been implemented as hypervisor extensions. The *AdCon* module for each server along with the *PriDyn* scheduler decides if a new request is accepted or rejected for scheduling on the system. A new application that is expected to cause performance degradation (for itself and/or for other applications already scheduled) on the system is identified in advance and is not accepted for execution on that system by *AdCon*. *PCOS* interacts with the *AdCon* modules on the servers and depending upon the outputs received from the modules, selects a server for scheduling every new request.

It is to be emphasized that the framework gives higher preference to the performance of currently executing I/O applications on the system since we want to avoid the number of VM migrations. As such, the *PCOS* framework will not schedule new requests on a system which is running high priority I/O applications. Therefore, the scheduling of a new request largely depends upon the state of the system apart from the latency sensitivity of the new request itself. Also, the aim of *PCOS* is not to attain *higher*, but *smarter* workload consolidation on the servers such that better utilization of disk resources can be achieved without compromising on performance. This will be elaborated in later sections of the thesis.

5.2.1 Design

Figure 5.6 shows an overview of the structure of the proposed framework with regard to its functionality on a single server that is replicated across multiple servers in the Cloud data-center. As mentioned before, the *Prescient Cloud I/O Scheduler* comprises of two main components

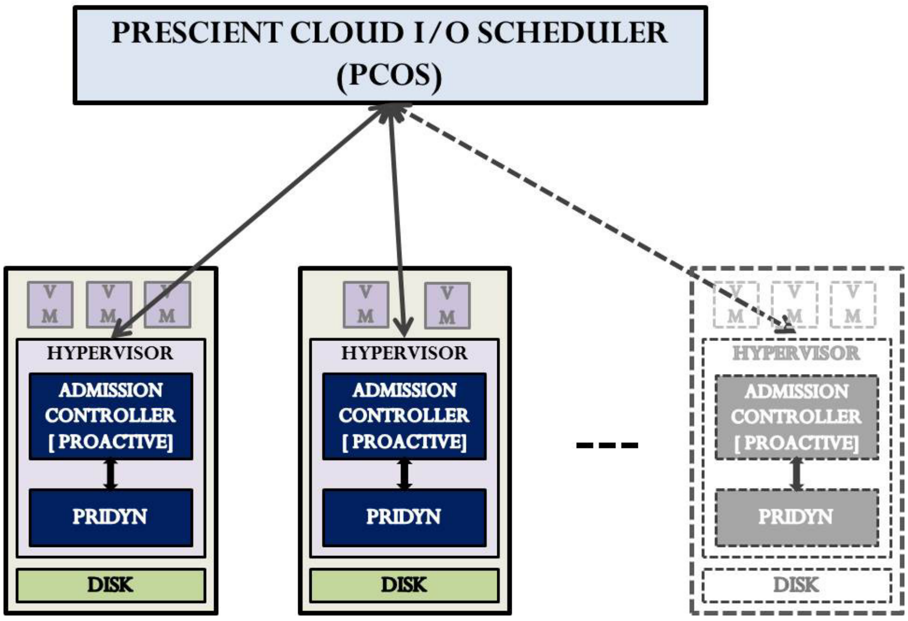

PRESCIENT CLOUD I/O SCHEDULER
(PCOS)
V M
HYPERVISOR
ADMISSION CONTROLLER [PROACTIVE]
PRIDYN
DISK

Figure 5.5: PCOS Framework

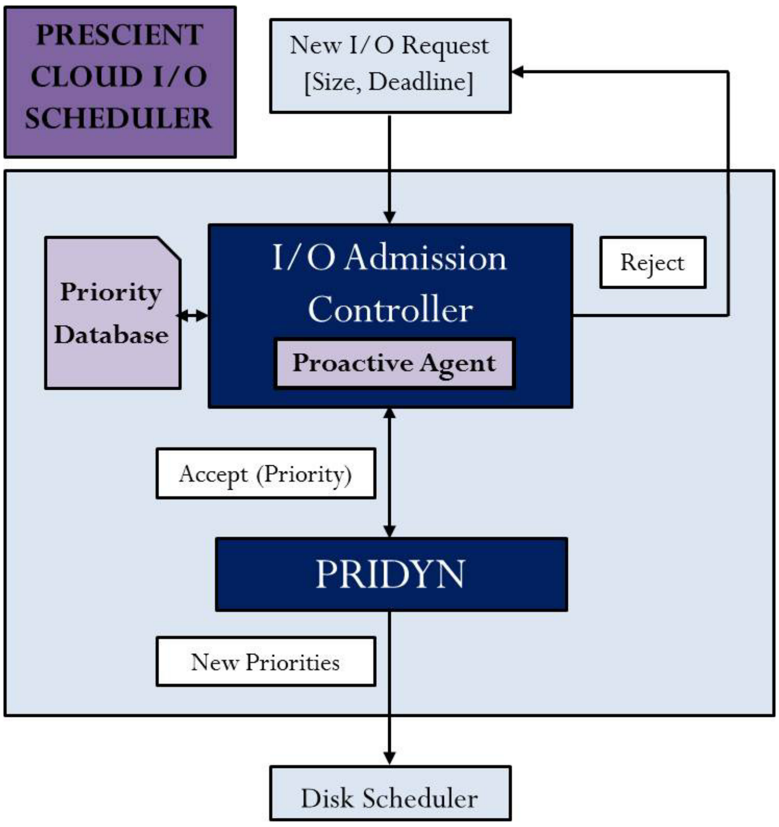


Figure 5.6: Structure of *PCOS*

running on the servers : The *Admission Controller (AdCon)* module and the *PriDyn* disk scheduler which work in conjunction with each other.

On each individual server, the *AdCon* module receives the information about the characteristics of a new I/O application request in terms of its size and deadline. At that time, *AdCon* collects information about the current state of resource allocation on the system in terms of the number of applications already running on the system along with their disk priorities with help of the *PriDyn* scheduler. It then anticipates the behavior of the system in the event of the new request being scheduled on the system before actually scheduling it. This is achieved by the *Proactive Agent* (part of *AdCon*) with the help of a *Priority Database.*

Priority Database : A local database stored on every individual server that stores information about the expected disk bandwidth allocation to applications depending upon the number and priorities of the applications, based on previous application execution history on that system. This is an iterative learning database that is continuously updated for different set of I/O applications having varying priorities, as and when new application combinations are scheduled on the system. If information about some specific combination is not present in the database, the nearest matching entry is used for estimation of bandwidth values and a new entry is added for future references. From this database, *AdCon* module gets an estimate of the performance of the applications according to the disk bandwidth that is expected to be allocated to them.

If the deadline of one or more applications is expected to be violated on assignment of default disk priority to the new request as per the estimate, *AdCon* takes the help of a modified version of *Priority Manager* module of *PriDyn* scheduler to find a suitable combination of disk priority values for the given set of applications that can satisfy the performance requirements of all the applications. If there exists such a combination of priority values which will cause no foreseeable deadline violations upon scheduling of the new request, the *AdCon* module accepts the request and it is scheduled on the system, otherwise the request is rejected for the current system state at that time. It is to be noted here that in *PriDyn*, if the priorities of the applications running on the system were changed by the *Priority Manager*, the modified priorities were actually implemented by the disk scheduler each time in order to observe application performance. However, in the current implementation, the priorities of the application are changed only theoretically by the modified version of *Priority Manager.* The *Priority Database* is used to predict application performance upon any proposed change in priorities without actual implementation. Only in the event that a request is accepted on the system with a modified priority set, the priority information is provided to the *PriDyn* scheduler which implements them through the physical disk scheduler on the system.

```
Require: $DataSize\ R_{new},\ Deadline\ D_{new}$
Ensure: $Server\ S_r\ for\ scheduling$
1: for $each\ server$ do
2:     $Call\ AdCon(R_{new}, D_{new})$
3:     if $Accept\ new$ then
4:         $Schedule\ new\ request$
5:     else
6:         $Continue$
7:     end if
8: end for
```

Figure 5.7: PCOS Algorithm

5.3 Implementation Details

In this section, we discuss the implementation details of the *PCOS* framework for a data-center with multiple servers. The algorithm for the main module of *PCOS* framework is shown in Figure 5.7. *PCOS* receives the information for every new I/O request in terms of its size R_{new} and deadline D_{new}. The algorithm is straightforward and involves calling the *AdCon* module for the servers for every new I/O request. The request is scheduled on a server S_R if it is accepted by the *AdCon* module on that server.

The implementation of the *AdCon* module on a server is explained in Figure 5.8. At a given instance, the number of applications running on the system is denoted by N. Every new request that is received from *PCOS* is denoted by $N+1$. The algorithm takes as input its datasize R_{N+1} and the desired deadline D_{N+1} and outputs the decision regarding the rejection or acceptance of $N+1$ request along with its priority Pr_{N+1}. Firstly, information is collected about each of the N running applications in terms of their data size R, deadline D, disk bandwidth B, start time of execution S and the disk priority Pr. The priority of the new application Pr_{N+1} is set to the default value at this point and it may be modified by the algorithm in subsequent steps if required. Next, it is assumed that the new application has already been scheduled on the system with the default priority and the behavior of the applications is anticipated in such a scenario. To foresee the expected latencies of the applications in the event that the new application was indeed scheduled, the sub module *Proactive Agent* is called for execution with the set of the priorities of all the applications (including the proposed new application) as input.

The functionality of *Proactive Agent* is explained in Figure 5.9. Upon receiving the set of application priorities, the algorithm accesses the *Priority Database* and searches for a potential match for the given priority set. If exact match is not found, then the closest match of the

```
Require: $DataSize\ R_{N+1},\ Deadline\ D_{N+1}$
Ensure: $Accept\ N+1\ (Pr_{N+1})\ or\ Reject\ N+1$
 1: $Find\ Current\ State\ (N, <R, D, B, S, Pr>)$
 2: $Set\ Pr_{N+1}\ to\ Default$
 3: $Call\ PROACTIVE\ AGENT(N+1, Pr_{<1...N+1>})$
 4: while (1) do
 5:     for $i\ in\ <1...N>$ do
 6:         $Find\ i\ s.t.\ L_i > (Di - (T - S_i))$
 7:     end for
 8:     if $(exists\ i)$ then
 9:         if $(L_{N+1} < (D_{N+1}))\ \&\ (Pr_{N+1} > lowest)$ then
10:             $Decrease\ Pr_{N+1}$
11:             $Call\ PROACTIVE\ AGENT(N+1,$
                                                  $Pr_{<1...N+1>})$
12:         else
13:             $Reject\ N+1$
14:         end if
15:     else                                       $\triangleright$ $deadlines\ met\ for\ all\ i\ in\ <1...N>$
16:         if $(L_{N+1} < (D_{N+1}))$ then
17:             $Accept\ N+1,\ (Pr_{N+1})$
18:         else
19:             $Call\ PRIORITY\ MANAGER(L_{<1...N+1>},$
                                                  $D_{<1...N+1>})$
20:         end if
21:     end if
22: end while
```

Figure 5.8: AdCon Algorithm

priority set (the number of applications necessarily matching) is selected and the expected values of disk bandwidth corresponding to the application priorities are recorded. The expected latencies of all the applications are then calculated by the *Latency Predictor* function of *PriDyn* scheduler with the help of disk bandwidth and total I/O size values. The implementation of the *Latency Predictor* is identical to that in *PriDyn* but the functionality is different. In case of *PriDyn*, the latencies of the applications were being calculated to provide differentiated I/O service but in *PCOS*, estimation of latencies of the applications is required for making meta-scheduling decisions. The set of latency values is returned to the main algorithm from the *Proactive Agent* sub module.

In the main module, the following steps are performed in a loop until a decision regarding the scheduling of the new request is made - for all N applications, the potential latency of each application is compared with the updated deadline value (Step 6) based on the time for which the application has already been executed (difference between the current time T and the start time S of the application). The algorithm also calculates whether the deadline of the new application $N + 1$ is being violated (note here the deadline value is the same as received in input since the application has not been actually scheduled yet). Depending upon the latency and deadline values, four possible cases are considered by this algorithm as following:

Case 1 : Deadline violated for one or more applications in $< 1 \ldots N >$*, deadline satisfied for* $N + 1$.

The priority of the new request is decreased if possible i.e. if the priority is not the lowest. The potential latencies of all applications are again calculated for new priority set with help of *Proactive Agent* (Step 11) in this case and the algorithm starts over from Step 5.

Case 2 : Deadline violated for one or more applications in $< 1 \ldots N >$*, deadline violated for* $N + 1$.

If the deadline of new application is also being violated in addition to other applications, the new request is rejected for the system at present state (Step 13). The state of the system i.e. the number and priority of the I/O applications is continuously tracked for any changes and the new request can again be considered for scheduling once the system state changes.

Case 3 : Deadline satisfied for all applications in $< 1 \ldots N >$*, deadline satisfied for* $N + 1$.

In the event that the deadlines of all the previously running applications are being satisfied after potentially scheduling the new request and the deadline of the new request is also being met, then the new request is accepted on the system with the assigned priority (Step 17). The priority of the new request will have the default value in case of the first iteration of the algorithm.

Case 4 : Deadline satisfied for all applications in $< 1 \ldots N >$*, deadline violated for* $N + 1$.

If the deadline of the new request is expected to be violated while all the other applications are estimated to meet their respective deadlines, then the algorithm tries to adjust the priorities of the applications in order to get a possible combination that suitably meets performance expectations of all the applications. For this, the modified *Priority Manager* sub module of the *PriDyn* framework module is called with the information about the latency and deadline values of all applications (Step 19).

The *Priority Manager* sub module (shown in Figure 5.10) will search among the N applications if there is such an application whose priority can be decreased (Step 2). If there is more than one such application, then the one having maximum gap between its expected latency and deadline value is chosen and its priority value is decreased (Step 5, 6). The latencies are recalculated by calling the *Proactive Agent* sub module (Step 7) and the algorithm begins new iteration. In the case that it is not possible to decrease the disk priority of any of the N

```
   $PROACTIVE\ AGENT(N+1, Pr_{<1...N+1>})$
 1: $Search\ Priority\ Database$
 2: $Update\ Bandwidth\ B_{<1...N+1>}$
 3: $Execute\ LATENCY\ PREDICTOR(R_{<1...N+1>},$
                                        $B_{<1...N+1>})$
 4: for $all\ i\ in < 1...N+1 >$ do
 5:     $RemainingData_i = R_i - DataProcessed_i$
 6:     $L_i = RemainingData_i / B_i$
 7: end for
 8: return $Latency\ L_{<1...N+1>}$
```

Figure 5.9: Proactive Agent

```
   $PRIORITY\ MANAGER(L_{<1...N+1>}, D_{<1...N+1>})$
 1: for $j\ in\ < 1...N >$ do
 2:     $Find\ all\ j\ s.t.\ (Pr_j > lowest)$
 3: end for
 4: if $(exists\ j)$ then
 5:     $Select\ j\ s.t.\ ((D_j - (T - S_j)) - L_j)\ is\ maximum$
 6:     $Decrease\ Pr_j$
 7:     $Call\ PROACTIVE\ AGENT(N+1, Pr_{<1...N+1>})$
 8: else
 9:     if $(Pr_{N+1} < highest)$ then
10:         $Increase\ Pr_{N+1}$
11:         $Call\ PROACTIVE\ AGENT(N+1,$
                                        $Pr_{<1...N+1>})$
12:     else
13:         $Reject\ N+1$
14:     end if
15: end if
16: return
```

Figure 5.10: Priority Manager

applications, the priority of the new request is increased (Step 10) in an attempt to meet its deadline value followed by the call to the *Proactive Agent* (Step 11). If all the above attempts fail, then the new process is rejected for the system in the current state (Step 13) and it will be checked again for scheduling once a new system state is obtained.

The *AdCon* algorithm is executed for every new I/O request that arrives on the given system. Only the application requests that are not detrimental for the overall system performance are admitted for execution on the system by the admission controller. The number of iterations of the algorithm required to arrive at a decision for a request will depend upon which of the above described cases holds true for the request. It has been made sure that the algorithm has a well-defined exit condition such that it necessarily finishes execution with a decision for every request within finite time. If a request is rejected for a given system state, it is tried again for scheduling on the system by the framework. Instead of continuous polling, *PCOS* has been optimized to keep track of the state of the systems such that the rejected request is reconsidered for scheduling on the same system by *PCOS* only when the state of the system changes, i.e. if the number or priority of the previously scheduled applications becomes different. This is done until the time for which the request has been under consideration for scheduling (scheduling delay) matches with its deadline value, at which point the request is considered as discarded and the next request is taken into consideration. In this manner, new I/O requests for the system can be continuously taken into consideration until they are either accepted for scheduling on the system or discarded completely. In the event that the request is discarded for the system, *PCOS* will iteratively consider other systems until the new I/O request is accepted for scheduling on a suitable system.

The time complexity of the algorithm on a server running N applications is simply of the order of N. In practical scenarios, the value of N cannot be arbitrarily large due to the physical limitations of the servers, thus keeping the complexity of the algorithm low. In the current study, we have analyzed the I/O behavior while operating on a local disk. In reality, the storage can be connected across the storage network. However, in either case it can be assumed that dedicated disk access is reasonable for workloads under consideration and the number of VMs contending for access to disk will be limited to a small number.

In the following section, the performance of the *PCOS* framework for ensuring a good mix of I/O applications for given systems is illustrated with the help of experiments on the real I/O traces.

5.4 Experimental Results

In this section, we explore the performance of *PCOS* as a meta-scheduler for Cloud data center. Experiments were performed for a single server on our setup and the observations can be generalized for multiple servers as well. Consider a server running two media applications concurrently on two separate VMs co-located on it, sharing the disk bandwidth. The deadlines for both the applications were assigned using the same value of delay tolerance factor δ for simplicity. In order to illustrate the functionality of *PCOS* as an admission controller, different kinds of new requests were considered for scheduling on the system. We consider the same workload combinations that were analyzed in Section 5.1.1. In the first case, the requests from a web server trace were given as input to the *PCOS* framework. Web applications are latency-sensitive since they are interactive applications and require small response times. The deadlines of these requests were therefore kept strict with small values of delay tolerance factor δ. The I/O requests for the web applications were considered for scheduling along with the two media applications on the system. The decision about accepting or rejecting a new I/O request depends upon the previously running applications, new requests will not be admitted if the applications executing on the system have strict deadlines. All experiments were performed for the same time duration for different values of δ for media applications in order to show the effect of strictness of deadlines of the previously scheduled applications on the number of new requests scheduled. Figure 5.11 shows various metrics (all values have been normalized) for the new application requests with different values of delay tolerance parameter δ. Measurements were done for the number of new I/O requests that were accepted for scheduling on the system by the *PCOS* framework as depicted by the green line. Also, the red line denotes the number of requests that were discarded i.e. the number of requests for which their deadline was reached while they were still under consideration for scheduling. As expected, the number of requests accepted on the system was higher for larger delay tolerance values since the deadlines were more lenient and more number of requests could be executed while still meeting the deadlines of previous applications. Also, the percentage of deadlines that were discarded became smaller as the value of delay tolerance parameter increased. Figure 5.11 also shows the values of Average Delay by the blue bars, which denotes the average time duration for which an application request had to wait from the time it was first considered for scheduling on the system till the time the request was finally accepted by *PCOS* for scheduling on system. This metric is valid only for the accepted requests that were actually scheduled on the system. It can be seen that the average delay time was higher for smaller values of delay tolerance parameter signifying that with stricter deadline values, new requests had to wait longer before being accepted on the system.

In the next set of experiments, the requests from a research server trace were scheduled on the system along with two media streaming applications. Since the requests for research application do not have strict deadlines, they have been assigned large value of delay tolerance factor δ (set to 20). Figure 5.12 shows that the number of research requests scheduled on the system increases with increasing values of delay tolerance parameter for media applications similar to the previous case. Also, the average waiting time for scheduling the requests decreases for higher delay tolerance parameter values as shown in the figure. Another observation is that no requests were discarded in this case unlike the previous case of web requests since the requests of research application were having smaller size and higher deadline values.

The most notable findings from these experiments are derived from the comparison between the two cases. Figure 5.13 shows the normalized values for the number of new requests accepted both for Case 1 and Case 2. The red bars denote the requests for the web application while the requests for the research application are shown by the blue bars. It can be seen from the figure that the number of requests scheduled for the web application requests is less than the number of research application requests scheduled on the server, for all the values of delay tolerance factor δ. Additionally, it was observed from the experimental results that the average waiting time for research requests was generally lower than that for web requests.

From these observations, it is clear that unlike other schedulers, *PCOS* performs scheduling of the application in a smart way such that new application requests are accepted on a server only when the performance requirements for all the applications, especially the previously running applications, are expected to be satisfied. If it is anticipated that the new requests will cause a performance degradation for the applications, the *Admission Controller* module of the *PCOS* framework does not accept the new requests to be scheduled on the server. Due to this reason, the number of web application requests which were accepted for the server was much lower than the number of research application requests as the web requests were expected to cause delays and deadline violations because of their latency sensitive characteristics. These findings clearly indicate that *PCOS* acts as an efficient admission controller for scheduling of applications on the system and ensures a good I/O workload combination. It ensures the scheduling of favorable combinations of I/O workloads on the servers and prohibits combinations which might lead to performance degradation.

Additionally, *PCOS* also ensures maximum utilization of disk resources while maintaining good application performance. Figure 5.14 shows the total disk utilization in terms of the disk bandwidth allocated to each application measured at the time intervals of execution of the *PCOS* algorithm when scheduling new research requests with media applications. The red and blue

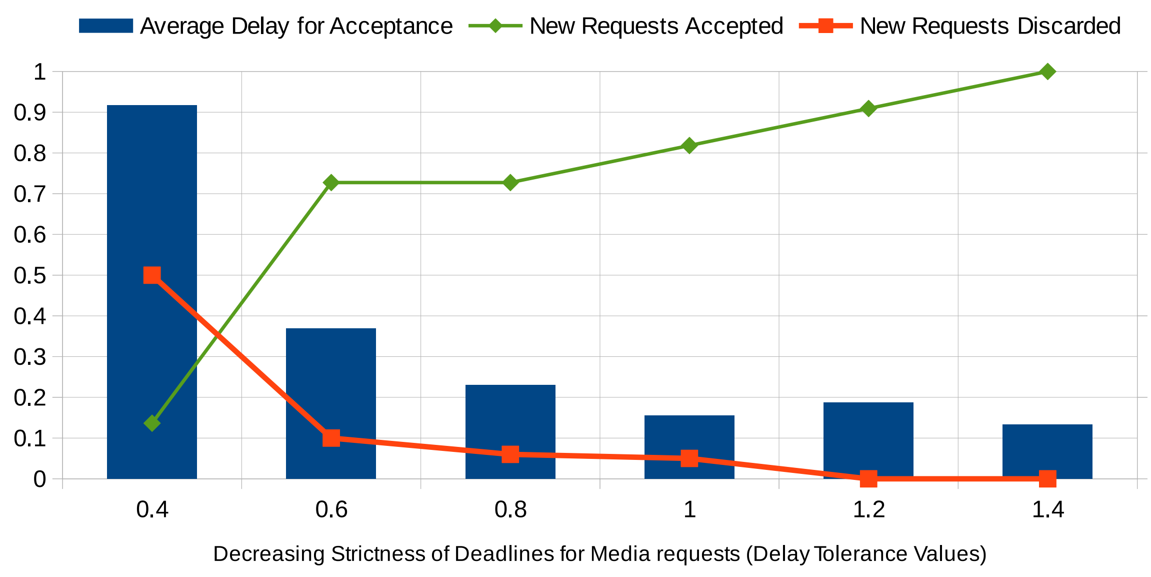


Figure 5.11: Performance of web requests with media applications (Case 1)

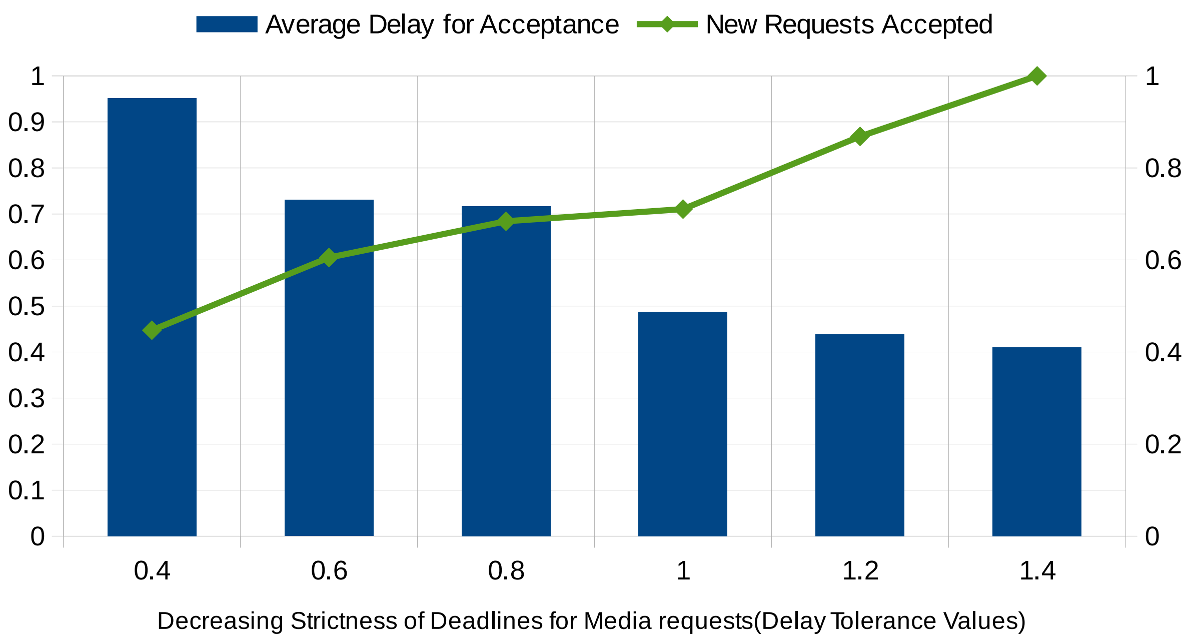


Figure 5.12: Performance of research requests with media applications (Case 2)

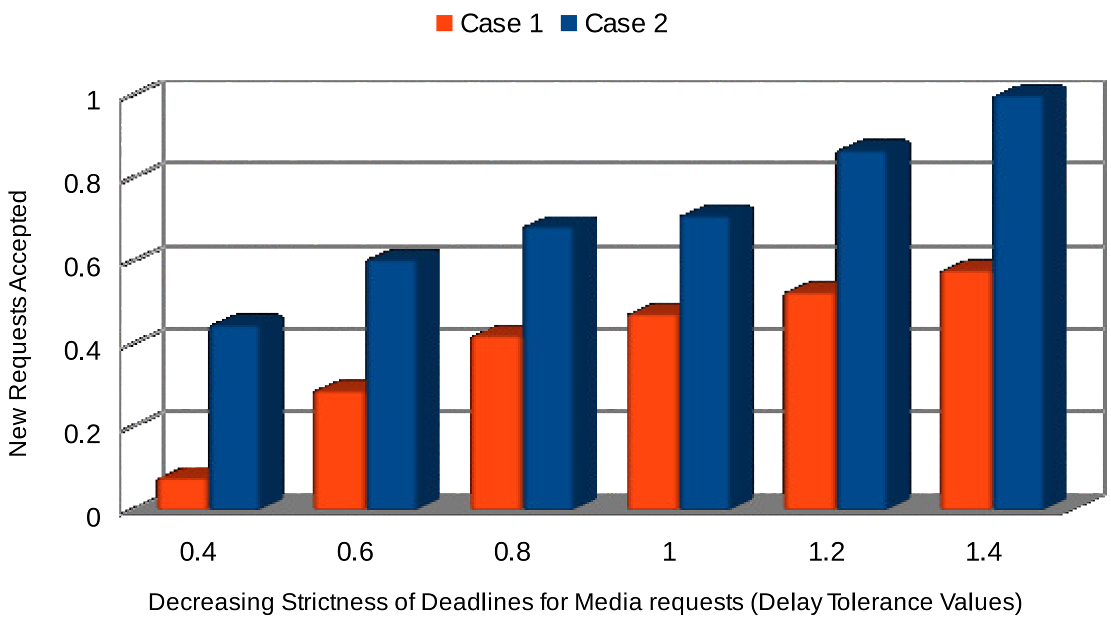


Figure 5.13: Comparison of number of requests scheduled in Case 1 and Case 2

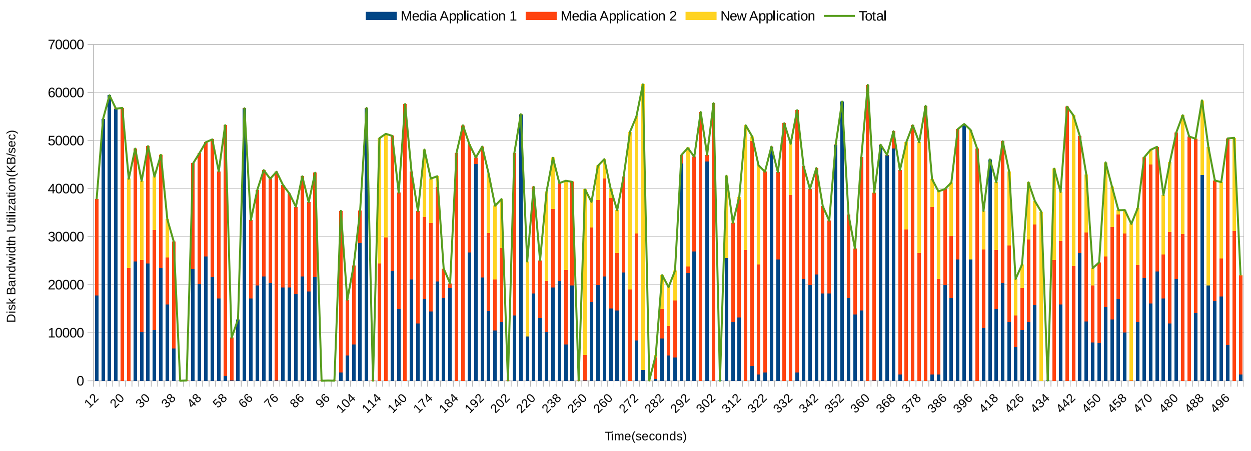


Figure 5.14: Total Disk Bandwidth Utilization

components of the stacked bars denote the disk bandwidth utilization of the media applications that were already executing on the server. It can be seen that the sum of the bandwidth utilization from the two media applications is much less than the total disk bandwidth achievable at any instance of time. The yellow bars indicate the bandwidth utilization of the new request from the research application. It is clear from the figure that the overall disk bandwidth utilization was increased by scheduling the research requests on the system signifying better resource utilization. Thus, it can be concluded from the observations made in this section that the *PCOS* framework can enable optimal placement of I/O workloads on a server for maximization of disk resources while also ensuring performance of applications. With the *PCOS* framework enabling optimum placement of applications for multiple servers, QoS based placement along with resource consolidation can be achieved at the data-center level.

5.5 Summary

In this chapter, we have motivated the need for a meta-scheduling framework for Cloud data-centers and proposed the design of the *PCOS* scheduling framework. By anticipating the expected performance of the applications on a server in advance, the *PCOS* framework can ensure that only those combinations of I/O workloads are scheduled on the server for which performance can be guaranteed. The algorithms for the implementation of the *PCOS* framework were described in detail to understand the functionality of the framework. Experimental analysis on real I/O workloads has demonstrated the utility and performance benefits that can be derived from the *PCOS* framework. Along with the *PriDyn* scheduler that dynamically updates disk priorities for the scheduled applications, the *PCOS* framework can achieve better server consolidation for I/O applications without compromising on application performance. In the next chapter, we present the conclusions of the current work and discuss future work planned in this direction.

Chapter 6

Conclusions and Future Work

In this chapter, we conclude the thesis with a brief discussion of the overall work. We discuss the problem statement regarding the degradation in performance for disk I/O workloads in virtualized data-centers. The encouraging results obtained with the help of our proposed scheduling frameworks justify that our approach was successful in achieving desired goals. This chapter also enlists the future work planned for enhancing the performance of the proposed frameworks to derive additional benefits in terms of energy-efficiency.

6.1 Conclusion

Virtualization enables consolidation of multiple applications on a single system to enable higher resource utilization. Being unaware of the degree of multi-tenancy, users running their application in virtualized Cloud environments expect the same performance as they would get if the application is run on a dedicated system. However, in the prevalent virtualization technologies, I/O workloads are affected due to sharing and this results in loss of performance for the applications. Failure to provide desired performance leads to SLA penalties for the Cloud provider. The current work explores the performance issues and the factors causing them in a comprehensive manner.

The issue of interference among workloads in a virtual setup is more complex than among normal user processes on a non-virtualized host due to the additional layer of abstraction that is associated with virtualization. The workloads running on individual VMs are oblivious of the requirements or characteristics of other workloads that are co-hosted with them and competing for the host resources. Also, each VM has its own software stack and scheduling policies that are independent of the actual physical resources and the scheduling policies of the host. Due to these reasons, performance interference is a pertinent problem for co-hosted workloads in

Cloud environments.

In this work, we have specially targeted I/O intensive workloads to bring into focus the I/O bottlenecks that are present in existing virtualization architectures. Hardware solutions like fast SSDs and SR-IOV architectures are available for high performance Cloud storage. But such architectures are not required or even suitable for general enterprise workloads where local disks on the host server are used for I/O purposes. In such scenarios, designing better scheduling techniques is the most desirable option to attain higher performance. Our approach is focused towards mitigating performance issues based upon existing hardware platform capabilities by affecting minimal changes to the current architectures.

For providing assured performance to enterprise workloads hosted in Cloud data-centers, it is essential to take into cognizance the characteristics of the workloads while deciding resource allocation policies. Accordingly, differentiated services need to be provided to the applications dynamically to meet QoS guarantees. For this purpose, we designed the *PriDyn* disk scheduler that is aware of the latency requirements of applications and dynamically allocates disk resources to achieve desired performance. Detailed evaluation of the framework has demonstrated that it can provide QoS guarantees to critical I/O applications by reducing latency to meet desired deadlines for execution. The framework also showed promising results on real I/O workloads where significant improvements were seen in overall system performance. This proves the utility of the proposed framework for real Cloud environments.

For a given set of applications, performance can be guaranteed by over-allocation of resources but such a technique will lead to wastage of resources. Server consolidation is an important aim of virtualization and this can be handled by intelligent VM placement and meta-scheduling strategies at the data-center level. This thesis proposed a novel admission controller and scheduling framework *PCOS*, as an extension to the *PriDyn* disk scheduler, which strives to achieve the balance between resource consolidation and application performance guarantees in Cloud environments. Working in conjunction, *PriDyn* and *PCOS* have been shown to achieve significant improvement in resource utilization enabling server consolidation without compromising on the performance desired by the applications for their functionality. Results derived from experiments performed on real world I/O traces have proved the efficacy of the proposed framework in choosing an optimal I/O workload combination to achieve the desired goals.

6.2 Future Work

Most of the existing disk schedulers have focused on fair-sharing or providing best-effort services to applications running in virtual setups. In this work, we have striven to achieve differentiated services to offer SLA guarantees to the applications for existing platforms. However, fine grained performance control to provide QoS guarantees will need significant changes to the architecture in terms of hardware support for virtualization. As observed in the results section, the proposed framework can extract good disk resource utilization while reducing deadline failures but it cannot guarantee satisfaction of all the deadlines. For delivering guaranteed services, participation of the physical device is necessary in resource allocation and placement strategies. As future work, we aim to test the proposed framework on architectures with hardware virtualization capabilities that are designed to support the features and benefits of the virtualization stack fully.

There are several other directions in which this work could be improved for enhanced performance. The *PCOS* framework was tested on a single system in this work, we plan to integrate the framework along with the scheduling mechanisms of widely used public Clouds to enable efficient data-center wide VM placement.

Further, the current framework does not take into account the energy requirements of the applications as a quantitative metric. For designing energy efficient data-centers, we plan to incorporate the power requirements of the applications in the *PCOS* framework in order to achieve optimal workload combinations in terms of energy efficiency for the data-centers. Such a scheduling framework will not only ensure application performance but also enable the design of 'green' data-centers.

6.3 Summary

This chapter concludes the thesis giving an overview of the present work. Concerns regarding the performance and QoS satisfaction of I/O intensive applications executing in virtualized setups is the main motivation for this work. We attempted to address this issue locally for every virtualized server through the *PriDyn* scheduler by providing differentiated services based on latency requirements. Further, to ensure workload consolidation and energy savings for Cloud data-centers, optimal workload placement and scheduling is achieved through the *PCOS* meta-scheduler. Overall, application performance and guaranteed QoS along with efficient resource utilization was demonstrated through the proposed frameworks. Related literature work and finally, directions for future work were also discussed in this chapter.

Bibliography

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A Berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009. (cited on page 1.)

[2] C. Waldspurger and M. Rosenblum, "I/O Virtualization," *Commun. ACM*, vol. 55, no. 1, pp. 66–73, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2063176.2063194 (cited on page 1.)

[3] R. McDougall and J. Anderson, "Virtualization Performance: Perspectives and Challenges Ahead," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 40–56, Dec. 2010. [Online]. Available: http://doi.acm.org/10.1145/1899928.1899933 (cited on page 1.)

[4] K. Adams and O. Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," *SIGARCH Comput. Archit. News*, vol. 34, no. 5, pp. 2–13, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1168919.1168860 (cited on page 1.)

[5] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the Gap Between Software and Hardware Techniques for I/O Virtualization," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404017 (cited on page 1.)

[6] N. Amit, M. Ben-Yehuda, D. Tsafrir, and A. Schuster, "vIOMMU: Efficient IOMMU Emulation," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002187 (cited on page 1.)

[7] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, "Towards High-quality I/O Virtualization," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 12:1–12:8. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534547 (cited on page 1.)

[8] "PCI SIG. I/O Virtualization," http://www.pcisig.com/specifications/iov/, [Online]. (cited on page 2.)

[9] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*. IEEE, 2009, pp. 50–57. (cited on page 3.)

[10] Amazon, http://aws.amazon.com/solutions/case-studies/netflix, [Online]. (cited on pages 3 and 19.)

[11] "Rackspace," http://www.rackspace.com/cloud/databases/, [Online]. (cited on page 3.)

[12] "DNANexus," https://cloud.google.com/files/DNANexus.pdf/, [Online]. (cited on page 3.)

[13] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/O performance of virtualized cloud environments," in *Proceedings of the second international workshop on Data intensive computing in the clouds*. ACM, 2011, pp. 71–80. (cited on pages 3 and 4.)

[14] "Enterprise Storage for Performance Driven Databases," https://labs.vmware.com/vmtj/virtualizing-latency-sensitive-applications-where-does-the-overhead-come-from, [Online]. (cited on page 3.)

[15] "Enterprise Storage for Performance Driven Databases," http://www.netapp.com/in/media/esg-lab-validation-ef-flash-arrays-for-oracle.pdf, [Online]. (cited on page 3.)

[16] I. Paul, S. Yalamanchili, and L. K. John, "Performance impact of virtual machine placement in a datacenter," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*. IEEE, 2012, pp. 424–431. (cited on page 3.)

[17] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 826–831. (cited on page 4.)

[18] S. L. Garfinkel, “Technical report tr-08-07: An evaluation of amazons grid computing services: EC2, S3 and SQS.” (cited on page 4.)

[19] R. Brown *et al.*, “US EPA ENERGY STAR Program, Report to congress on server and data center energy efficiency: Public law 109-431,” *Lawrence Berkeley National Laboratory*, 2008. (cited on page 4.)

[20] “NRDC,” http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf, [Online]. (cited on page 4.)

[21] Y. Ding, X. Qin, L. Liu, and T. Wang, “Energy efficient scheduling of virtual machines in cloud with deadline constraint ,” *Future Generation Computer Systems*, no. 0, pp. –, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15000369 (cited on page 4.)

[22] A. Gulati, C. Kumar, and I. Ahmad, “Storage workload characterization and consolidation in virtualized environments,” in *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009. (cited on pages 4 and 16.)

[23] Y. Kanemasa, Q. Wang, J. Li, M. Matsubara, and C. Pu, “Revisiting Performance Interference Among Consolidated n-Tier Applications: Sharing is Better Than Isolation,” in *Proceedings of the 2013 IEEE International Conference on Services Computing*, ser. SCC ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 136–143. [Online]. Available: http://dx.doi.org/10.1109/SCC.2013.42 (cited on page 4.)

[24] “OpenStack,” https://www.openstack.org/, [Online]. (cited on page 5.)

[25] J.-T. Tsai, J.-C. Fang, and J.-H. Chou, “Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm ,” *Computers and Operations Research*, vol. 40, no. 12, pp. 3045 – 3055, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030505481300169X (cited on page 5.)

[26] Y. Goto, “Kernel-based Virtual Machine Technology,” *Fujitsu Sci. Tech. J*, vol. 47, no. 3, pp. 362–368, 2011. (cited on page 5.)

[27] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the linux virtual machine monitor,” in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230. (cited on page 5.)

[28] "IOZone," http://www.iozone.org/, [Online]. (cited on page 5.)

[29] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46. (cited on page 6.)

[30] "OProfile," http://oprofile.sourceforge.net/, [Online]. (cited on page 6.)

[31] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on.* IEEE, 2007, pp. 200–209. (cited on pages 14 and 16.)

[32] J. Shafer, "I/O virtualization bottlenecks in cloud computing today," in *Proceedings of the 2nd conference on I/O virtualization.* USENIX Association, 2010, pp. 5–5. (cited on page 14.)

[33] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz, "Disk scheduling with quality of service guarantees," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 2. IEEE, 1999, pp. 400–405. (cited on page 14.)

[34] R. Wijayaratne and A. N. Reddy, "Integrated QOS management for disk I/O," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 487–492. (cited on page 14.)

[35] D. Li, X. Liao, H. Jin, B. Zhou, and Q. Zhang, "A new disk I/O model of virtualized cloud environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1129–1138, 2013. (cited on page 14.)

[36] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passé?" *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 20–24, 2010. (cited on page 15.)

[37] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk I/O scheduling in virtual machines," in *Proceedings of the 2nd conference on I/O virtualization.* USENIX Association, 2010, pp. 6–6. (cited on page 15.)

[38] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments.* ACM, 2008, pp. 1–10. (cited on page 15.)

[39] L. Cherkasova, D. Gupta, and A. Vahdat, "When virtual is harder than real: Resource allocation challenges in virtual machine based it environments," *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-25*, 2007. (cited on page 15.)

[40] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003. (cited on page 15.)

[41] X. Lin, Y. Mao, F. Li, and R. Ricci, "Towards fair sharing of block storage in a multi-tenant cloud," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012, pp. 15–15. (cited on page 15.)

[42] S. Kim, D. Kang, and J. Choi, "Fine-grained I/O Fairness Analysis in Virtualized Environments," in *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, ser. RACS '12. New York, NY, USA: ACM, 2012, pp. 403–408. [Online]. Available: http://doi.acm.org/10.1145/2401603.2401690 (cited on page 15.)

[43] S. R. Seelam and P. J. Teller, "Virtual I/O scheduler: A scheduler of schedulers for performance virtualization," in *Proceedings of the 3rd international conference on Virtual execution environments*. ACM, 2007, pp. 105–115. (cited on page 15.)

[44] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: Performance Insulation for Shared Storage Servers." in *FAST*, vol. 7, 2007, pp. 5–5. (cited on page 15.)

[45] J. Zhang, A. Sivasubramaniam, Q. Wang, A. Riska, and E. Riedel, "Storage performance virtualization via throughput and latency control," *ACM Transactions on Storage (TOS)*, vol. 2, no. 3, pp. 283–308, 2006. (cited on page 15.)

[46] A. Gulati, A. Merchant, and P. J. Varman, "pClock: an arrival curve based approach for QoS guarantees in shared storage systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 13–24, 2007. (cited on page 15.)

[47] A. Gulati, I. Ahmad, C. A. Waldspurger *et al.*, "PARDA: Proportional Allocation of Resources for Distributed Storage Access," in *FAST*, vol. 9, 2009, pp. 85–98. (cited on page 15.)

[48] A. Gulati, A. Merchant, and P. J. Varman, "mClock: handling throughput variability for hypervisor IO scheduling," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–7. (cited on page 15.)

[49] P. J. Shenoy and H. M. Vin, “Cello: a disk scheduling framework for next generation operating systems,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1. ACM, 1998, pp. 44–55. (cited on page 15.)

[50] D. Shue, M. J. Freedman, and A. Shaikh, “Performance Isolation and Fairness for Multi-tenant Cloud Storage,” in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 349–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387914 (cited on page 15.)

[51] D.-J. Kang, C.-Y. Kim, K.-H. Kim, and S.-I. Jung, “Proportional disk I/O bandwidth management for server virtualization environment,” in *Computer Science and Information Technology, 2008. ICCSIT’08. International Conference on.* IEEE, 2008, pp. 647–653. (cited on page 15.)

[52] Y. Sfakianakis, S. Mavridis, A. Papagiannis, S. Papageorgiou, M. Fountoulakis, M. Marazakis, and A. Bilas, “Vanguard: Increasing Server Efficiency via Workload Isolation in the Storage I/O Path,” in *Proceedings of the ACM Symposium on Cloud Computing.* ACM, 2014, pp. 1–13. (cited on page 15.)

[53] M. Kesavan, A. Gavrilovska, and K. Schwan, “Differential virtual time (DVT): Rethinking I/O service differentiation for virtual machines,” in *Proceedings of the 1st ACM symposium on Cloud computing.* ACM, 2010, pp. 27–38. (cited on page 15.)

[54] G. Casale, S. Kraft, and D. Krishnamurthy, “A model of storage I/O performance interference in virtualized systems,” in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on.* IEEE, 2011, pp. 34–39. (cited on page 16.)

[55] I. Ahmad, J. M. Anderson, A. M. Holler, R. Kambo, and V. Makhija, “An analysis of disk performance in VMware ESX server virtual machines,” in *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on.* IEEE, 2003, pp. 65–76. (cited on page 16.)

[56] I. Ahmad, “Easy and efficient disk I/O workload characterization in VMware ESX server,” in *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on.* IEEE, 2007, pp. 149–158. (cited on page 16.)

[57] R. N. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds," in *Web Information Systems Engineering-WISE 2012*. Springer, 2012, pp. 171–184. (cited on pages 17 and 38.)

[58] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya, "SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter," in *Algorithms and Architectures for Parallel Processing*. Springer, 2011, pp. 371–384. (cited on page 17.)

[59] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: ACM, 2013, pp. 77–88. [Online]. Available: http://doi.acm.org/10.1145/2451116.2451125 (cited on page 17.)

[60] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, "Pesto: Online storage performance management in virtualized datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 19. (cited on page 17.)

[61] A. Gulati, C. Kumar, and I. Ahmad, "Modeling workloads and devices for IO load balancing in virtualized environments," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 61–66, 2010. (cited on page 17.)

[62] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "BASIL: Automated IO Load Balancing Across Storage Devices," in *FAST*, vol. 10, 2010. (cited on page 17.)

[63] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes, "Minerva: An automated resource provisioning tool for large-scale storage systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 19, no. 4, pp. 483–518, 2001. (cited on page 17.)

[64] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. C. Veitch, "Hippodrome: Running Circles Around Storage Administration," in *FAST*, vol. 2, 2002, pp. 175–188. (cited on page 17.)

[65] A. Gulati and I. Ahmad, "Towards distributed storage resource management using flow control," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 10–16, 2008. (cited on page 17.)

[66] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," in *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 219–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=2535461.2535489 (cited on page 17.)

[67] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 237–250. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755938 (cited on page 17.)

[68] "iotop," http://linux.die.net/man/1/iotop, [Online]. (cited on page 23.)

[69] J. Axboe, "Time Sliced CFQ I/O Scheduler," http://kerneltrap.org/node/4406, [Online]. (cited on page 25.)

[70] "Snia," http://www.iotta.snia.org/, [Online]. (cited on page 35.)

[71] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008. (cited on page 35.)

[72] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013. (cited on page 38.)

[73] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 22. (cited on page 38.)

[74] R. Calheiros and R. Buyya, "Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication," 2013. (cited on page 38.)

[75] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 228–235. (cited on page 38.)

[76] A. Strunk, "Costs of virtual machine live migration: A survey," in *Services (SERVICES), 2012 IEEE Eighth World Congress on.* IEEE, 2012, pp. 323–329. (cited on page 45.)