MINERVA: Proactive Disk Scheduling for QoS in Multi-Tier, Multi-Tenant Cloud Environments

Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara, *Members, IEEE*

Abstract—In recent years, internet-connected devices have become ubiquitous, leading to increased demand for computing and storage resources. The cloud has emerged as a cost-effective and scalable solution to these infrastructure requirements, making it possible to consolidate several disparate computing resources into a single physical machine. However, while the availability of multi-core CPUs, high-capacity memory modules, and virtualization technologies has increased the density of *converged infrastructure*, disk I/O has remained a substantial performance bottleneck, especially when dealing with high-capacity mechanical disks. In this study, we investigate how proactive disk scheduling, backed by predictive models and client-side coordination, can influence the overall throughput and responsiveness of a cluster in data-intensive computing environments. We evaluate our framework with a representative MapReduce job on a 1,200-VM cluster, demonstrating a 21% improvement in completion time under heavy disk contention.

Index Terms—Scheduling and task partitioning, Distributed architectures, Performance attributes

1 INTRODUCTION

THERE has been significant growth in stored data volumes with the proliferation of networked sensing equipment, mobile devices, web services, and social media applications. Public and private clouds manage a considerable percentage of this information, with data-intensive applications benefiting from the elastic scalability of virtualized deployments. These applications often involve frequent use of disk I/O with both sequential and random operations underpinning analytics tasks.

Cloud settings provide excellent service isolation and packaging convenience. In many cases, cloud providers over-provision physical hardware to optimize for costs and real-world usage patterns. Modern hypervisors are able to handle CPU and memory contention due to over-provisioning, but I/O contention poses a particularly difficult challenge due to the electro-mechanical nature of hard disk drives that are frequently used in big data applications. While solid state drives (SSDs) provide better performance than HDDs, I/O costs are still orders of magnitude higher than memory accesses.

Disk scheduling inefficiencies have adverse impacts on turnaround times and throughput for analysis operations, which are often implemented as MapReduce jobs. These inefficiencies can lead to higher contention and introduce performance bottlenecks. The primary focus of this study is to support proactive disk I/O scheduling across virtual machines with the objective of increasing overall throughput, ensuring long-term fairness, and reducing turnaround times. Our framework, MINERVA, proactively alleviates disk contention, amortizes I/O costs, and selectively prioritizes VMs based on access patterns and durations.

- *This research was supported by funding from the US Department of Homeland Security (HSHQDC-13-C-B0018 and D15PC00279), the US National Science Foundation (CNS-1253908), and the Environmental Defense Fund (0164-000000-10410-100).*

1.1 Challenges

Proactive disk scheduling in virtualized settings introduces unique challenges, including:

- ***Long-term Fairness***: Rather than focusing on short-term fairness, I/O history and patterns must be considered to determine scheduling priorities.
- ***Proactive Scheduling***: Addressing issues in a reactive fashion can have adverse consequences and reduce throughput, especially if I/O durations are short.
- ***Managing Diversity***: Throughput and fairness must be maintained across disparate users, I/O patterns, and resource consumption profiles.
- ***Transparency***: While VMs may participate in the scheduling process, knowledge of the framework should not be required to benefit from it.

1.2 Research Questions

We explored several research questions in this study that guided the design of Minerva:

1) What externally observable VM characteristics can be used to inform disk scheduling?
2) How can we identify I/O patterns? This involves predicting not just when burst patterns are likely to occur, but also the type and duration of the patterns.
3) How can we facilitate proactive scheduling? This may include predictive and time-series models.
4) How can we ensure that scheduling decisions improve system throughput and response times for collocated processes?
5) How can we incorporate support for VMs to influence scheduling decisions? Specifically, how can we enable interactions with Minerva through client-side libraries?

1.3 Paper Contributions

Minerva provides a means for allocating resources fairly on a long-term basis across processing entities (VMs, containers, and processes). This helps mitigate issues such as *noisy neighbors* that consume more than their fair share of resources or employ disruptive I/O patterns. We accomplish this by (1) allowing and incentivizing participation from processing entities, (2) tracking and harvesting observable characteristics of these clients, (3) constructing models to forecast expected I/O patterns, and (4) making alternative storage options available to amortize I/O costs and reduce contention. These factors are used to determine I/O *weights* that are passed to the underlying operating system to influence resource allocations and scheduling priorities.

1.4 Related Work

Kernel disk schedulers generally strive for *short-term fairness* between processes competing for I/O resources, where each process gets an equal share of the disk. This can be achieved with round robin scheduling or by assigning each process a time quantum to use the disk. Other approaches consider physical properties of disks, such as rotational media [1] or flash-based storage [2]. The primary difference between Minerva and traditional disk schedulers is our goal of achieving *long-term fairness* to account for processes that use a disproportionate amount of resources. Minerva is not a disk scheduler; rather, it employs user-space daemons to gain insight about systems it manages to influence the underlying OS scheduler.

Systems like VM-PSQ [3] address I/O prioritization in the context of virtualization by acting as an agent for VM I/O. These approaches employ *backend* and *frontend* drivers to coordinate operations between the host system and virtual machines. Queuing weights are used to determine VM priorities, which allow QoS rules to be implemented for different types of VMs. While Minerva also assigns I/O weights to both processes and VMs, its primary focus is on *long-term fairness* aided by **monitoring** and **predicting** disk access patterns and differences in I/O classes.

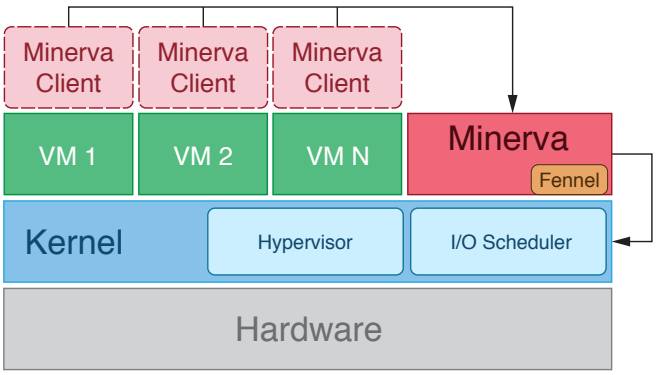

Fig. 1. Minerva architecture: Minerva uses the *Fennel* framework to dispatch scheduling directives and collect information about hardware and processes running on the host machine, while Minerva client applications running in VMs or containers can optionally provide fine-grained information about their state to improve coordination and prediction of I/O contention.

Several approaches incorporate prediction of I/O workloads and access patterns to improve the effectiveness of schedulers or prefetching algorithms. These include profiling applications on reference hardware and then scaling benchmark results to predict process completion times [4], or using predictive models to determine the probability of future I/O events [5]. While many approaches focus on offline modeling, Minerva uses **online modeling** to adapt to resource utilization trends at run time.

2 Minerva: System Architecture

The Minerva framework consists of three core components that work together to improve I/O performance:

- *Fennel* [6], a low-level disk prioritization agent.
- *Minerva server*, which runs at each physical machine and monitors system metrics related to I/O
- *Minerva clients* inside VMs or containers

Figure 1 illustrates the flow of communication between these components and how they interact with the underlying OS. Note that VMs or containers that do not include a Minerva client are still monitored and influenced by the server; the client library is not required to benefit from framework, but applications that coordinate with it can achieve better performance.

2.1 Fennel

The Fennel daemon [6] provides dynamic I/O prioritization in distributed and data-intensive computing environments. While most disk schedulers attempt to fairly balance access to resources, Fennel prioritizes virtual machines and processes that make short, concise I/O requests. In other words, this strategy rewards processes that use disk resources sparingly. Fennel leverages the Linux kernel *Control Groups* functionality to dynamically assign disk priorities. While Fennel does not depend on a particular disk scheduler, our implementation uses the Linux CFQ scheduler as it is the only scheduler that currently implements I/O weighting.

Fennel places VMs or processes into resource groups that allow the system to assign dynamic scheduling weights, define unique rules, and classify collections of processes beyond their individual lifetimes. Each group is assigned a block I/O *weight* that determines its share of the overall disk throughput. As a process uses more or less of a resource, its priority changes dynamically. The default Fennel configuration creates 10 priority groups and uses the total number of bytes read or written on a moving average to determine resource utilization. Fennel does not require modification (or any participation) of guest VMs or processes, and is a *reactive* strategy. On the other hand, Minerva leverages the Fennel framework along with system resource models to *proactively* mitigate resource contention before it occurs. In this study, we compare new functionality in Minerva with the Fennel framework to demonstrate how our enhancements improve upon Fennel.

2.2 Minerva Server

The Minerva server extends Fennel to influence scheduling decisions. It also maintains information about host machines, client processes, VMs, and containers that allow it

to build resource utilization models and predict future I/O interactions. This information is gathered both internally by the framework and externally, and includes metadata provided by clients through requests issued from the Minerva *client library*. This library allows applications to explicitly state (or estimate) their usage patterns and requirements upfront, which helps improve the scheduling decisions made by the system. A single Minerva server instance is run on each host machine, and communicates with clients through TCP over the internal network. Scheduling decisions are made on a per-disk basis, and metadata collected by the server is periodically persisted to disk in case of failures or planned outages.

2.3 Minerva Client

On the client side, Minerva operates as both an application library and user-space daemon. No changes to VMs or containers are required to act as a Minerva client; instead, applications that use the library automatically start the client daemon, which is responsible for reporting disk utilization and system metrics to the server. Most cloud providers allow communication through the internal private network, so we perform resource discovery and management over a TCP interface. To further ease integration with our framework, we provide a set of command-line utilities that implement Minerva functionality. These tools are lightweight, and give administrators a simple means to communicate with the Minerva server; for instance, executing `minerva-ctl priority low` would tell the scheduling agent that the I/O priority for the particular VM can be decreased in relation to other clients.

2.4 Experimental Setup

Benchmarks in the following sections were run on an overprovisioned private cloud of 1,200 virtual machines. Each VM was allocated a CPU core, 512-1024 MB of RAM, and a virtual disk image for storage. Host machines were configured with Fedora 21 and the KVM hypervisor, with guest VMs running a mixture of Fedora 21 and Ubuntu Server 14 LTS. Each physical host machine was equipped with four hard disks, dual gigabit interfaces, and included 45 HP DL160 servers (Xeon E5620, 12 GB RAM) and 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM).

3 Endogenic Scheduling

OS disk schedulers are tasked with allocating resources, and often rely on *externally observable* (exogenic) traits to determine both *how* and *when* processes use the disks. Factors may include physical properties such as the amount of disk head movements required or the spatial locality of requests, as well as system constraints such as process priorities. Minerva extends this concept to include *internal* (endogenic) traits as well, making processes, containers, and virtual machines part of the scheduling process. We provide two mechanisms that allow VMs to participate in reducing contention: *voluntary deprioritization* and *temporally-coordinated scheduling*. In both cases, processes are given incentives to encourage participation, but are not penalized for operating outside the framework.

3.1 Voluntary Deprioritization

For situations where an application needs to perform maintenance operations or execute background tasks, Minerva encourages processes to request *voluntary deprioritization*. This notifies Minerva that the process can be scheduled at a lower priority, influenced by an optional scheduling level: `low`, `lowest`, and `idle`, where the `idle` level signals that the process should only be scheduled when no other process is using the disk (however, this level does not guarantee liveness). Voluntary deprioritization is especially useful when processes on a given host machine are servicing different timezones; VMs can request that they be deprioritized during nighttime hours (or other low-traffic periods), giving other VMs a greater share of I/O resources.

To incentivize voluntary deprioritization, processes accumulate *scheduling credits* that allow them to request a higher scheduling priority at a later point in time. For example, a video streaming service might elect to run at the lowest priority during the night, a low priority during working hours, and then use the accumulated scheduling credits to boost performance for the deluge of requests that occurs when users arrive home in the evening. Scheduling credits are earned and spent based on time slices (one second by default), and are weighted based on the I/O priority and *contention level* of the system.

Minerva calculates the contention level by totaling the number of processes actively using the disk at a given time slice, and adds a small bonus based on the process priority level. Our default configuration assigns a weight of 1.0 to the contention level and 0.10 to the scheduling priority. This approach assigns more scheduling credits to processes that are deprioritized when the disk is in high demand, while still allocating a small reward to processes that voluntarily lower their priority in times of low contention. To use scheduling credits, client applications *purchase* time slices where they will be scheduled at a high priority. The cost of the credits and when they expire is configured by administrators, and our default implementation ships with the cost of a high-priority time slice being set to 25 scheduling credits with expiration occurring after one day. This configuration results in a deprioritized task earning about 2.5 minutes of high priority scheduling per hour when there is little or no contention, and 14 minutes per hour when four processes are accessing the same disk.

3.2 Temporally-Coordinated Scheduling

Another function Minerva provides to clients is *temporally-coordinated* scheduling. In this access pattern, clients ask the scheduler for guidance on when they should carry out their I/O operations. If disk contention is significant, processes may be requested to delay I/O for short periods of time in the interest of letting other tasks complete first. By coordinating with the scheduler, VMs can improve perceived disk performance and reduce overall contention.

To initiate temporally-coordinated scheduling, client applications send a request to the Minerva server asking for an approximate *scheduling time*. The request can contain additional metadata about the nature of the I/O that will be performed, including the size of the request and an optional deadline that the scheduler will use to avoid starvation.

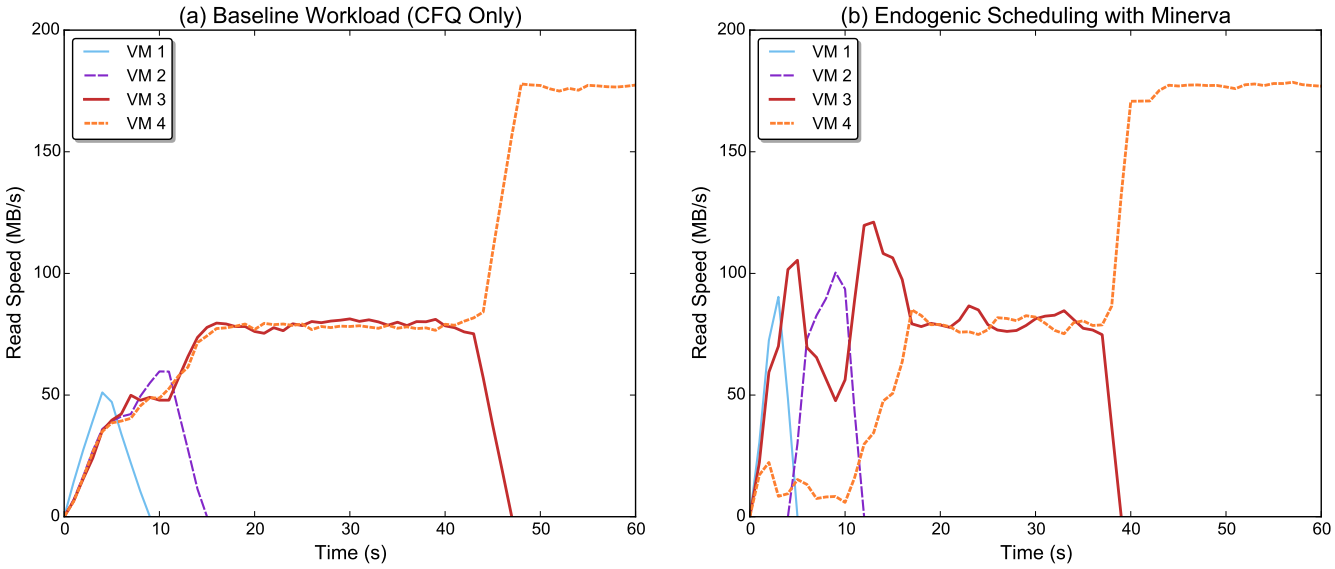

Fig. 2. Performance evaluation of endogenic scheduling with four workloads. By coordinating with Minerva, VMs 1, 2, and 3 achieve significantly higher read speeds; endogenic scheduling reduced I/O times for each workload by 29%, 38%, 13%, and 1%, respectively (across VMs 1 – 4).

After making the request, the server responds by instructing the client to (1) perform the operation immediately, (2) wait for a number of time slices, or (3) wait for a 'ready' callback notification. As a further benefit to the client, the scheduler makes use of the request metadata to determine how long the operation should take and then calculates the average I/O priority of the process ahead of time. This improves the consistency of the disk throughput observed by the client application. In situations where another process or VM begins a disk operation at the same time as a temporally-coordinated I/O event, the coordinated client receives a higher overall scheduling weight to offset resource contention from the non-coordinated process.

3.3 Endogenic Performance Evaluation

To evaluate how endogenic scheduling can improve performance under I/O contention, we launched four disk-intensive tasks at the same time in separate VMs:

1) Verifying the checksum of 200 MB of data
2) Loading a 500 MB machine learning dataset
3) Hadoop *grep* job scanning 3 GB of log files
4) Hadoop *word count* job processing 12 GB of ebooks

Figure 2-a illustrates the read speed of these tasks using the CFQ scheduler, which ensures fairness but results in poor overall performance for VMs 1 and 2. Since the tasks are started concurrently, a history-based weighting system such as Fennel would be unable to differentiate between the VMs. On the other hand, Figure 2-b highlights the effects of endogenic scheduling provided by Minerva, where the client application in VM 4 notifies the scheduler that it does not need high-priority access to the disk (voluntary deprioritization), and VM 2 requests a callback to do its 500 MB read within the next 20 seconds. This approach boosts the observed read speed for VMs 1, 2, and 3, while also rewarding VM 4 with scheduling credits that can be used at a later point in time. Note that VM 3 does not coordinate with Minerva in this benchmark, but still benefits from its scheduling decisions. Additionally, VMs 3 and 4 reach scheduling parity as VM 3 consumes more resources over time.

4 Proactive I/O Scheduling

Dynamically adjusting scheduling weights based on observed system metrics improves performance by balancing access to I/O resources. However, this approach is *reactive*; the scheduler is only able to calculate proper request weighting after observing the processes. This is sufficient in many cases, but also results in situations where the scheduler does not have enough information to distinguish between processes. To improve scheduling decisions, we provide *proactive* I/O modeling to predict future disk operations. This is achieved with two models: short-term forecasting of I/O events before they occur with *gradient boosting*, and long-term time-series analysis to help discover patterns in disk usage with *ARIMA*.

4.1 Forecasting I/O Events

Each instance of the Minerva server maintains resource usage models of the VMs and processes it manages. Our first model, implemented with gradient boosting, predicts the magnitude of future disk events by inspecting system metrics that correspond with I/O requests. Gradient boosting is an ensemble method that creates multiple base prediction models (decision trees, in our case) and then iteratively boosts their performance to create a composite model with low prediction bias. We train the model with data collected at each time slice, along with the magnitude of I/O (in MB/s) that occurred in the following time slice (if any). Metrics we use as training data include:

- Percent CPU usage
- Memory consumption
- Network throughput

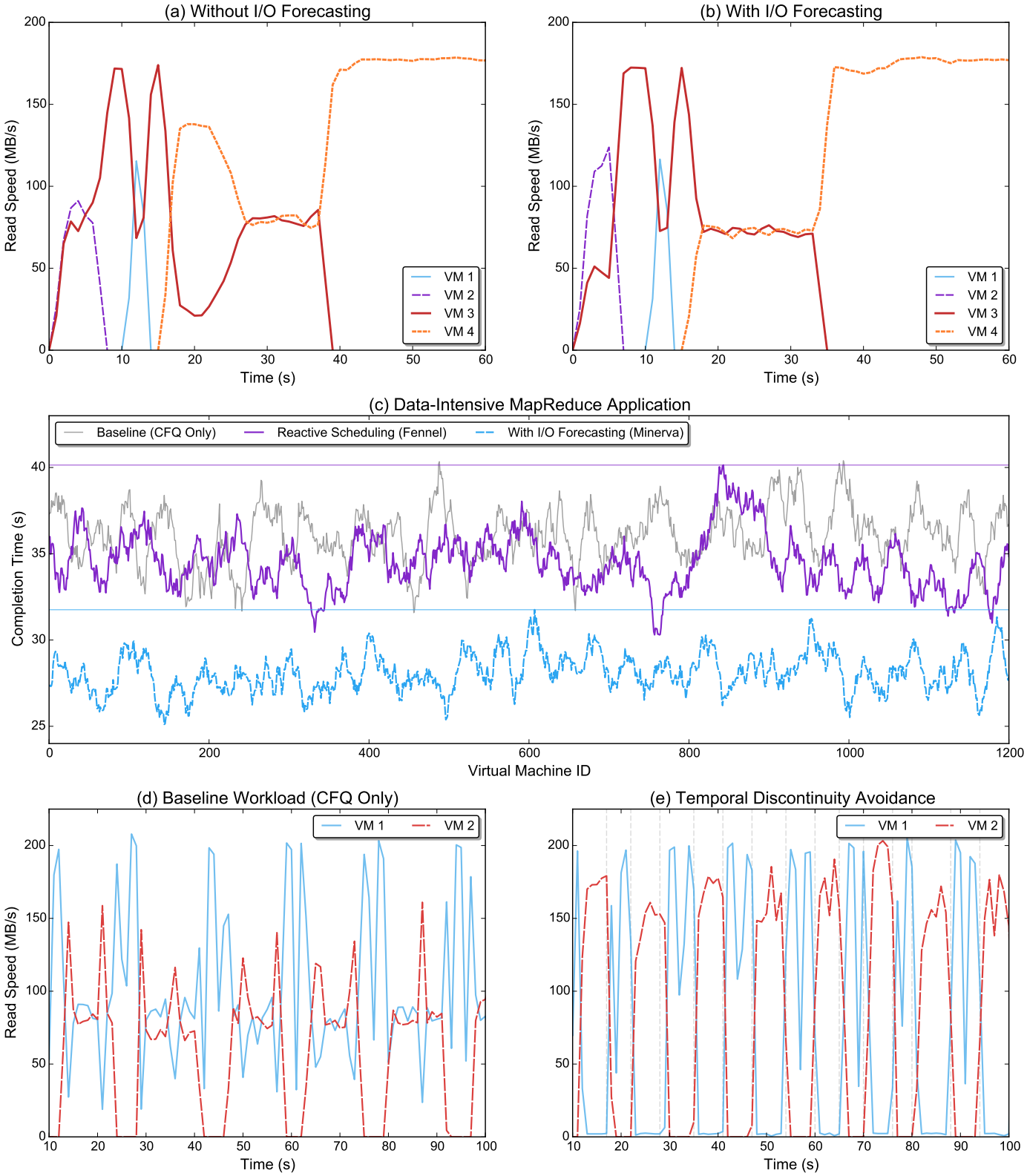

Fig. 3. Proactive I/O scheduling benchmarks: (a) VM workloads weighted based on historical usage (Fennel) compared with (b) proactive weighting provided by I/O forecasts, (c) a demonstration of a MapReduce application running across 1200 VMs (denoted by VM IDs) with and without I/O forecasting from our gradient boosted models to circumvent disk contention (total job run time indicated by horizontal lines), (d) illustration of I/O contention and scheduling discontinuities between VMs, (discontinuities are evident when both processes are operating at a low read speed) and (e) temporal discontinuity avoidance with our ARIMA models to interleave I/O operations from two different workloads (dashed lines indicate scheduling priority changes in response to model outputs).

We also record the number of open file descriptors, size of open files, and the number of incoming and outgoing network connections, if available. Since virtual machines appear to be a single, monolithic process to the host OS, we divide training data samples into *I/O sessions* to distinguish between events. These sessions are delineated by a new file being opened, a sharp increase in memory consumption, or additional network connections being opened.

In our test environment, I/O magnitude prediction resulted in forecasts with a root-mean-square deviation of about 15 MB/s — enough accuracy to predict the I/O class of a new I/O session. Minerva also continuously evaluates the quality of its predictions to adjust how much they impact starting priorities. Figure 3-a demonstrates a scenario that benefits from proactive I/O weighting (using the VM workloads described in the previous section). Initially, the reactive approach is unable to differentiate between VMs 2 and 3, and later favors VM 4 over 3 even though they are both heavy users of the disk. Conversely, Minerva is able to forecast starting I/O weights, as shown in Figure 3-b. Figure 3-c illustrates how data-intensive MapReduce applications can benefit from I/O forecasting; in this benchmark we launched a Hadoop *word count* application to analyze and histogram approximately 2 TB of book content, while also assimilating 10 TB of atmospheric data into a Galileo [7] distributed storage framework instance spanning the 1,200 VMs in our test cluster. With Minerva enabled, the heavy I/O patterns exhibited by Galileo were detected and deprioritized within 5 seconds, yielding resources to the MapReduce application. In this evaluation, proactive scheduling resulted in a 21% improvement in the overall run time of the MapReduce application, with an 18% reduction in per-task average run times.

4.2 Temporal Discontinuity Avoidance

Another aspect of proactive disk scheduling is using time-series models to discover I/O patterns. Many applications follow discernible trends when it comes to I/O; for instance, an application may read data, process it, and then flush the results to stable storage. Such patterns provide opportunities to schedule other disk activities during processing or dormant phases. To forecast future I/O patterns, we use autoregressive integrated moving average (ARIMA) models on the observed I/O rates (both read and write) for each process. ARIMA models are particularly useful for non-stationary data that may exhibit trends or seasonal fluctuations. We forecast 15 seconds into the future and use the Hyndman-Khandakar algorithm to autonomously parameterize these models.

To evaluate the effectiveness of our long-term time-series ARIMA models, we benchmarked two virtual machines performing competing I/O tasks. In this test, VM 1 was responsible for executing a data integrity check that involved reading files from the disk, checksumming them with a MD5 hash, and then verifying that their checksum matched a known good value. On the other hand, VM 2 was tasked with running a Galileo storage node instance that was responsible for receiving new observations over the network, processing them, and then writing the resulting data blocks to disk. These two tasks lead to *scheduling discontinuities*, where the disk may be idle when both VMs are processing information. To alleviate scheduling discontinuities, Minerva considers the temporal patterns associated with processes as well as their resource consumption profiles to better interleave I/O activities. Figure 3-d illustrates how the workload at VM 1 tends to receive a disproportionate amount of I/O resources over time, even though it pauses its I/O activities frequently to calculate checksums. On the other hand, Figure 3-e demonstrates how Minerva is able to detect the I/O patterns in both applications to equalize their share of the disk while also avoiding scheduling discontinuities.

5 Multi-Tier Storage Cache

Minerva provides several tools that enable VMs, processes, and host machines to suppress resource contention. However, some processes, such as distributed file systems, are unable to avoid issuing large amounts of I/O requests. To manage these scenarios, we provide a multi-tier storage service to help optimize for use cases that involve frequent disk accesses. This approach gives client applications access to a shared pool of memory and stable storage that is managed by the Minerva server. In other words, applications gain the ability to use resources outside their usual allocations, and the Minerva server can route high-throughput I/O to hardware optimized for the task, such as solid state drives (SSDs).

The *Minerva Cache* is made available through our client library as well as a FUSE file system (Filesystem in User Space). Applications are presented with an isolated directory tree that can be mounted and manipulated with standard file system interfaces. There are two root storage directories in a Minerva file system: *temporary* and *stable* storage. The temporary storage pool resides in main memory and is designed for small files that may be lost in the event of a hardware or power failure. In situations where the host machine does not have enough free memory for caching, temporary files will be persisted to the stable storage pool. Stable storage is designed for files or critical data that must be preserved in the event of a system failure. In general, files in stable storage should be written to an SSD or high-RPM mechanical disk that is not allocated directly to any client processes or VMs.

To further incentivize the use of our stable storage pool, Minerva provides *delayed migration* functionality. In this storage model, files are persisted to stable storage through a specific directory in the Minerva FUSE file system and then *symlinked* to a corresponding location on the local file system. This allows the file to be accessed from the local file system as if it were stored there. During periods of low disk contention, the Minerva client daemon migrates these cached files to the local file system, updates symlinks, and then removes the file from the cache. This approach enables clients to amortize the cost of writing to their own file systems and also helps Minerva reduce overall I/O contention. On a server equipped with an SSD in our test deployment, clients were able to write to the temporary and stable pools at 900 MB/s and 422 MB/s, respectively, with read speeds of 1500 MB/s and 468 MB/s with 1 GB of data (averaged over 100 iterations).

6 Conclusions

Proactively scheduling VM I/O bursts by monitoring and forecasting disk usage patterns achieves long-term fairness across hosted VMs. Since I/O usage is host-specific and dependent on the mix and behavior of VMs, offline approaches tend to be unsuitable. Minerva relies on online models trained with host-specific resource usage data to better capture usage patterns at a particular host. Time-series analysis of access patterns is key to identifying scheduling discontinuities. Once identified, scheduling discontinuities can be alleviated through prioritization and scheduling decisions. In our 1200-VM textual analysis MapReduce experiment, alleviating contention improved job completion time by 21% with an 18% reduction in per-task run times. Further, we involve VMs, containers, and processes in the scheduling framework through voluntary deprioritizations, temporally coordinated scheduling, and a multi-tier storage and caching layer to help avoid and subvert disk contention.

References

[1] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O," *SIGOPS Oper. Syst. Rev.*, pp. 117–130, 2001.

[2] S. Park and K. Shen, "FIOS: A fair, efficient flash I/O scheduler," in *Proc. of the 10th USENIX Conference on File and Storage Technologies*. USENIX Association, 2012.

[3] 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, pp. 647–653.

[4] M. Meswani, P. Cicotti, J. He, and A. Snavely, "Predicting disk I/O time of HPC applications on flash drives," in *GLOBECOM Workshops (GC Wkshps)*, 2010, pp. 1926–1929.

[5] J. Oly and D. A. Reed, "Markov model prediction of I/O requests for scientific applications," in *Proceedings of the 16th International Conference on Supercomputing*. ACM, 2002.

[6] M. Malensek, S. L. Pallickara, and S. Pallickara, "Alleviation of disk I/O contention in virtualized settings for data-intensive computing," in *(To Appear) Proceedings of the 2015 IEEE/ACM International Symposium on Big Data Computing*.

[7] M. Malensek, S. Pallickara, and S. Pallickara, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Generation Computer Systems*, 2012.

Matthew Malensek is a Ph.D. student in the Department of Computer Science at Colorado State University. His research involves the design and implementation of large-scale distributed systems, data-intensive computing, and cloud computing. Matthew received his Masters degree in Computer Science from Colorado State University. Email: malensek@cs.colostate.edu

Sangmi Lee Pallickara is an Assistant Professor in the Department of Computer Science at Colorado State University. Her research interests are in the area of large-scale scientific data management, data mining, scientific metadata, and data-intensive computing. She received her Masters and Ph.D. degrees in Computer Science from Syracuse University and Florida State University, respectively. Email: sangmi@cs.colostate.edu

Shrideep Pallickara is an Associate Professor in the Department of Computer Science at Colorado State University. His research interests are in the area of large-scale distributed systems, specifically cloud computing and streaming. He received his Masters and Ph.D. degrees from Syracuse University. He is a recipient of an NSF CAREER award. Email: shrideep@cs.colostate.edu