# Alleviation of Disk I/O Contention in Virtualized Settings for Data-Intensive Computing

Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara
Department of Computer Science
Colorado State University, Fort Collins, Colorado, USA
Email: {malensek, sangmi, shrideep}@cs.colostate.edu

***Abstract*—Steady growth in storage and processing capabilities has led to the accumulation of large-scale datasets that contain valuable insight into the interactions of complex systems, long- and short-term trends, and real-world phenomena. *Converged infrastructure*, operating on cloud deployments and private clusters, has emerged as an energy-efficient and cost-effective means of coping with these computing demands. However, increased collocation of storage and processing activities often leads to greater contention for resources in high-use situations. This issue is particularly pronounced when running distributed computations (such as MapReduce applications), because overall execution times are dependent on the completion time of the slowest task(s).**

**In this study, we propose a framework that makes opinionated disk scheduling decisions to ensure high throughput for tasks that use I/O resources conservatively, while still maintaining the average performance of long-running batch processing operations. Our solution does not require modification of client applications or virtual machines, and we illustrate its efficacy on a cluster of 1,200 VMs with a variety of datasets that span over 1 Petabyte of information; in situations with high disk interference, our algorithm resulted in a 25% improvement in MapReduce completion times.**

***Index Terms*—Data-intensive computing, I/O interference, big data, distributed I/O performance**

## I. Introduction

Stored data volumes have grown at a remarkable rate due to the proliferation of sensors, observational equipment, mobile devices, e-commerce, and social networks. A study from IDC suggests that storage requirements will breach 40 ZB by 2020, with cloud and big data analytics solutions growing three times faster than on-premise solutions [1]. These analytics jobs, typically expressed as MapReduce computations [2], operate on voluminous datasets that span multiple disks. The most popular software stack for achieving this is the Apache Hadoop [3] ecosystem that includes the core Hadoop MapReduce framework, HDFS for scalable storage [4], Hive for data summarization [5], and Pig [6] for expressing analysis tasks using high-level constructs.

Growth in data volumes has coincided with improvements in stable storage, available CPU cores, and memory capacities. Increased densities and capacities of stable storage, both electromechanical hard disk drives (HDDs) and solid state drives (SSDs), allow ever greater data volumes to be stored at a single node. These factors have led to increases in the number of processes that concurrently execute on a given node, which in turn increases the degree of contention for resources.

Disk I/O contention incurs higher costs in terms of latency and throughput than either CPU or memory contention, and adversely impacts completion times and throughput for data-intensive tasks. This stems from the nature of the underlying storage technology and associated seek times of HDDs, which are six orders of magnitude slower than CPU clock cycles. With mechanical disks, processes often contend for a single disk head. Stable storage must also satisfy different performance objectives; for example, in the case of HDDs the objective of disk scheduling algorithms is to minimize disk head movements, while in the case of SSDs the objective is to minimize write amplifications and make considerations for wear leveling. In both cases, buffering helps increase throughput while minimizing disk head movements and write amplifications.

Data-intensive analytics tasks are typically performed in public or private clouds where resources are provisioned as Virtual Machines (VMs). VMs offer excellent containment and isolation properties while coping with issues relating to simplifying software dependencies, and can be scaled elastically as demand fluctuates.

### A. Research Challenges

Our research objective for this work is to ensure high-throughput completion of data-intensive tasks in both virtualized and non-virtualized settings while ensuring fairness across processes and VMs. Challenges involved in accomplishing this include:

- ***Applicability***: no changes must be made to the virtual machines or client processes to benefit from our framework. We must avoid requiring client-side software libraries, message passing, or custom kernels.
- ***Over-provisioning***: one of the key features of cloud deployments is the ability to run several services on a single machine under loose resource provisioning constraints. However, over-provisioning also increases contention between collocated VMs.
- ***Fairness***: private and public clouds often involve multiple, concurrent analytics jobs being executed at the same time at any given physical machine. These jobs may have different mixes of I/O and computing activities. High throughput processing should not be achieved at the expense of fairness.

### B. Research Questions

Key research questions that we explore in this study include:

1) How can we minimize I/O contention between collocated VMs?
2) How can we detect and avoid disk access patterns that are known to reduce overall I/O throughput?
3) How can we ensure fairness across a set of analytic tasks? This involves profiling the mix of client disk I/O and processing activities and then making appropriate scheduling decisions to prioritize certain processes.
4) How can we achieve high throughput across a cluster while avoiding resource starvation?

This paper includes empirical evaluations that profile the efficiency of our proposed framework to address these research questions. We also compare and contrast with the current state-of-the-art in our literature survey in Section II. Often, deficient approaches: (1) require modifications to the VMs, with inclusion of device drivers to boost the priority of favored VMs, (2) involve custom kernels at the hypervisor level, or (3) require client software or libraries to be packaged alongside the analytic tasks. None of these approaches take a holistic view necessary for orchestrating analytics jobs that target alleviating disk I/O interference at any given node while accounting for the mix of tasks that execute at a particular physical machine. In data-intensive computing, job completion times are often determined by the slowest constituent tasks.

### C. Approach Summary

Our framework, ***Fennel***, runs as a system daemon and aims to reduce disk I/O interference and ensure high-throughput processing. This involves: (1) profiling VMs to identify their disk usage patterns, (2) creating of resource group classes to reflect priority levels, (3) dynamically adjusting these priority levels based on VM profiles to influence decisions made by the native disk I/O scheduler. In this study we work with the Linux kernel and the corresponding VMs and containers that execute in Linux clusters; Linux is the dominant OS used in cloud settings and accounts for 75% of the installations in data centers [7]. Figure 1 illustrates where Fennel exists in the hardware/software stack: the framework operates as a user space daemon and communicates with the OS disk scheduler through kernel interfaces.

Rather than striving to be fair to all processes accessing the disk at a given point in time, our algorithm introduces a scheduling bias towards applications that use the disk infrequently or in short, concise bursts. This approach boosts the performance of applications that read or write relatively small amounts of data on a periodic basis, while still preserving the long-term average performance for applications that use the disk frequently or deal with large files. We use the Linux *cgroup* functionality to create "resource classes." Our default configuration divides processes into 10 weight classes, with 10 representing the highest priority and 1 representing the lowest priority. Our framework then uses the Linux *CFQ* scheduler [8] to dynamically tune process (and process group) disk scheduling priorities at run time.

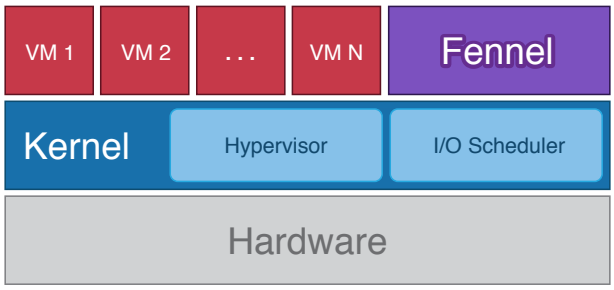


Fig. 1. An overview of the Fennel system architecture. Fennel operates independently above the OS kernel and disk scheduler as a user space application, passing dynamic scheduling directives to the kernel layer based on interactions and requirements of client software (VMs or other processes).

A related aspect of ensuring fairness is how long to penalize a process for its disk usage. If a process slows down or pauses its disk use, its priority will slowly increase at a configurable rate. In our default configuration, each time quantum that passes with no I/O activity from a process decreases its cumulative disk I/O counter by 50% of the increment used for decreasing the priority. This ensures that several collocated processes or VMs performing similar amounts of disk I/O will have a similar priority.

We run an instance of the Fennel daemon at each physical machine. The scheduler polls the system resource counters for disk accesses on a configurable interval (once per second by default). This returns the processes that have performed I/O during the last second. The daemon manages resource counters that accumulate as processes perform I/O operations, and then adjusts their scheduling weights based on a configurable set of rules. After inspecting the current I/O state, Fennel removes records of processes that have terminated and updates the weights of tasks that have not performed disk operations during the previous quantum, if applicable.

Our empirical benchmarks profile the performance of our framework in virtualized settings with data-intensive tasks executed using Hadoop. Our benchmarks report on the performance with and without our framework, as well as in single-machine and distributed settings. We have performed this evaluation with a mix of jobs running within the cluster, and were able to improve MapReduce task completion times by 25% in a scenario with high disk contention.

### D. Paper Contributions

This paper demonstrates the feasibility of reducing disk I/O interference in virtualized environments while preserving high-throughput completion of data-intensive jobs and tasks. Specifically, our contributions include:

- Dynamic tuning of priority levels based on usage patterns.
- Detection of disk access patterns that are detrimental to overall throughput. Specifically, access patterns that perform excessive seek operations on rotational media will be penalized.
- The ability to ensure fairness across processes without compromising on throughput.

- Fennel is broadly applicable and does not require modifications to the VM or the hypervisor kernel, inclusion of device drivers, or custom APIs.
- Changes to priority levels are constant-time operations and bookkeeping data structures used to track I/O usage patterns are efficient.
- Fennel reduces the number of stragglers in MapReduce jobs and lowers the amount of speculative tasks that will be launched at run time.

### E. Experimental Setup

The benchmarks carried out in this study were run on a heterogeneous, over-provisioned private cloud consisting of 1,200 VMs. Each VM was allocated a single processor core, 512-1024 MB of RAM, access to a bonded gigabit network link, and a virtual hard disk for storage. Host machines ran Fedora 21 and the KVM hypervisor, with guest VMs running a mixture of Fedora 21 and Ubuntu Server 14 LTS. Host machines included 40 HP DL160 servers (Xeon E5620, 12 GB RAM), 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM), and 5 HP Z210 workstations (Xeon E31230, 12 GB RAM). Each physical machine in the cluster was equipped with four disks.

We used a variety of datasets to benchmark the performance of our framework in a data-intensive computing environment. This included over 100 TB of website data from the Common Crawl [9], 10 TB of system logs collected from 300 workstations in a lab environment, raw feature data and graphical tiles generated from the NOAA North American Mesoscale Forecast System (NAM) [10] totaling 1 Petabyte.

### F. Paper Organization

The rest of the paper is organized as follows. Section II outlines related work in disk I/O schedulers and other types of scheduling control schemes. Section III describes how disk scheduling is implemented in the Linux kernel, and provides insight into the different types of tradeoffs the various schedulers manage. Section IV provides a detailed overview of the functionality and components in Fennel, followed by Section V with a thorough performance evaluation of our algorithms in both single-machine and distributed settings. Finally, we conclude the paper and outline our future research directions in Section VI.

## II. Related Work

Considerable research has been conducted on operating system disk I/O scheduling, including simple first-come, first-served (FCFS) queuing approaches, rotationally-aware algorithms [11], [12], [8], and, most recently, schedulers that specifically target flash-based disks [13], [14], [15]. Section III provides a detailed overview of current scheduling technologies and the trade-offs associated with them; our framework operates above the I/O scheduler level and is broadly applicable across scheduling implementations, provided that they include a weighted prioritization mechanism. One primary differentiator of Fennel is its concept of *fairness*: while a fair scheduler may ensure that each process gets a share of the disk at a given point in time, Fennel targets *long-term fairness* where processes that use a disproportionate amount of disk resources overall are assigned a lower priority than others.

VM-PSQ [16] addresses I/O prioritization in the context of virtualization by acting as an agent for VM I/O that is separate from the system disk scheduler. In this approach, queues are created for each VM and then a *backend driver* on the host machine uses a token scheduling algorithm to distribute *time slices* to each VM based on configurable token weights. Client VMs communicate with the backend driver through a *frontend driver* that coordinates I/O operations. Since VM-PSQ focuses on VM I/O instead of the entire system, it is able to achieve lower variance in disk performance between competing VMs when compared to the default Linux scheduler (CFQ [8]). While our approach also assigns I/O weights, it operates above the scheduling layer and is designed to dynamically address long-term fairness rather than strict short-term fairness.

Similar to VM-PSQ, IO QoS [17] employs a back-end/frontend driver scheme to intercept I/O operations from guest virtual machines and then places the requests into per-VM queues. It uses a combination of the leaky bucket algorithm [18] to control I/O speeds and weighted fair queuing (WFQ) [19] to prioritize different virtual machines based on user QoS (quality of service) requirements. Both VM-PSQ and IO QoS are designed strictly for virtual machines, whereas our approach targets I/O across all processes on a system.

Virtual I/O Scheduler (VIOS) [20] is a "scheduler of schedulers" that assigns each application a quantum of time for accessing the disk. This constraint acts as a hard limit on per-process I/O, which ensures fairness at a coarse-grained level. To provide fine-grained fairness or QoS for particular processes (including virtual machines), VIOS also allows a second I/O scheduler to be assigned on a per-application basis. This fine-grained scheduler can be selected based on application requirements or hardware characteristics. For hardware that supports tagged command queuing (TCQ) or native command queuing (NCQ), the VIOS(P) variant of the scheduler supports issuing multiple commands to the disk controller in parallel, improving performance. VIOS could be used as the underlying scheduler in Fennel by either implementing a Linux control groups interface or using Fennel interfaces to communicate with the scheduler directly.

Range-BW [21] combines the proportional scheduling of VM-PSQ with a Linux device-mapper driver, *dm-ioband*, to provide predictable I/O bandwidth. To facilitate predictability, the desired minimum and maximum bandwidth can be specified for processes and changed at run time. Similar to our framework, Range-BW does not depend on a particular scheduler implementation and employs Linux control groups for managing groups of processes. On the other hand, Fennel is designed to modify and update scheduling priorities autonomously at runtime without prior knowledge of the workloads it will manage (which may also follow temporal trends or change as different virtual machines or applications are deployed to the cluster).

## III. Linux Disk Scheduling

The latest Linux kernel (version 4.1.3 at the time of writing) includes three disk I/O schedulers: *noop*, *deadline*, and *CFQ* (Completely Fair Queueing [8]). CFQ is currently the default scheduler for most Linux distributions, supplanting the previous default, *Anticipatory Scheduling* (AS). In general, disk schedulers improve I/O performance by reordering and batching requests before they are sent to the underlying hardware; if requests are transferred immediately to the disk controller, excessive seeking could occur in the case of mechanical disks, or processes may suffer from I/O starvation. The active scheduling algorithm can be changed at run time on a per-device basis to account for hardware characteristics and variations in workload requirements.

While using a disk scheduler is beneficial in most cases, there are some exceptions. These instances are handled by the *noop* (no operation) scheduler, which queues and merges incoming requests before transferring them directly to the underlying hardware. This minimalist scheduling approach is most useful in scenarios where the hardware has large caches, built-in support for scheduling, or disk configurations that cannot be optimized for at the operating system level. Virtual machine instances that contain a limited number of client applications can also benefit from using the noop scheduler, essentially offloading scheduling concerns to the hypervisor or host operating system. However, VMs that must maintain fairness among their own processes will likely require a more advanced scheduler [22]. Solid state disks may exhibit improved performance when paired with the noop scheduler, but workloads that involve disk contention are generally best served by the deadline or CFQ schedulers.

The *deadline* scheduler is primarily focused on latency, and prevents I/O starvation by enforcing a deadline for request start times. Since applications are more likely to block while performing read operations (the data being read is often required to continue processing), reads are prioritized over writes. Operations are sorted and processed in batches based on their associated logical block addresses (LBAs), and separate *deadline queues* are inspected after each batch has completed processing to ensure no requests have been starved. The deadline scheduler is useful in situations that involve multi-threaded workloads [23] or small, random reads combined with sequential buffered writes (frequently observed in databases) [24].

*Anticipatory Scheduling* (AS) [12] focuses on dealing with *deceptive idleness*, a phenomena that occurs when processes appear to be finished reading from the disk but are actually processing data to prepare for the next read operation. If deceptive idleness is not accounted for, unnecessary seek operations become more likely as other processes' incoming requests differ greatly in physical location on disk. In these situations, AS allows the disk to stay in an idle state for short periods of time after servicing a read request in anticipation of further requests — an optimization that resulted in a 29-71% throughput improvement in disk-intensive Apache HTTP server workloads [12]. While AS may increase the amount of time a disk spends idling, exploiting spatial locality in the I/O requests results in net performance gains. As one might expect, anticipatory scheduling requires some tuning to account for differences in disk characteristics; an idle time that works well with mechanical disks may result in an unfair distribution of resources on solid state disks [25].

The current default Linux disk scheduler, *Completely Fair Queuing* (CFQ), builds on the concept of *stochastic fairness queueing* proposed by McKenney [26]. Fairness queuing is implemented by mapping each process, network source, or other type of traffic flow to queues that are served on a round-robin basis. This ensures that each traffic flow will receive an equal share of the resource, but also takes some processing time to map flows to queues. Stochastic fairness queuing reduces these computational requirements by creating a fixed number of queues and then assigning the flows by passing source-destination address pairs through a hash function. Unfortunately, this approach suffers from potential collisions, especially when the number of disk requests are much larger than the number of queues available.

CFQ avoids collisions by creating a queue for each thread group in the system and then distributes I/O resources evenly across the queues. It also implements a short pause after servicing each queue to deal with deceptive idleness, and allows the pause to be tuned (or disabled completely) by system administrators to address the differences in mechanical and solid state disks. Somewhat similar to the deadline scheduler, CFQ offers a low-latency mode that allows a *target latency* to be specified when servicing requests that favors latency over throughput. By placing I/O requests into individual queues, CFQ makes fine-grained prioritization of different processes and thread groups possible. We leverage this functionality in our framework to penalize processes that use a disproportionate amount of I/O resources and improve the overall long-term throughput of virtual machines running on a given system. I/O prioritization is exposed through the Linux kernel *control group* (cgroup) interface.

### A. Control Groups and I/O Classes

In the Linux kernel, *Control Groups* (often abbreviated as *cgroups*) are a unified interface for managing system resources. Each type of resource is managed through corresponding *subsystems*, which include CPU, memory, network, and block I/O resource controllers. Control groups allow hierarchical rules to be applied to certain classes of processes or users, as well as *OS-level virtualization* which provides complete namespace isolation for groups of processes. OS-level virtualization functionality in Linux has been used in projects such as Linux Containers (LXC) [27], [28] and Docker [29]. Our framework uses the block I/O subsystem to manage I/O priorities. There are three system I/O classes:

- **Idle**: processes running at this priority will receive disk access only when no other processes are using the disk.
- **Best-effort**: the default class, which can be tuned with a sub-priority ranging from 0-7. The default sub-priority is calculated as $sp = (nice + 20)/5$, where *nice* refers to the CPU scheduling priority of the process.
- **Realtime**: processes are given first priority access to the disk. May result in starvation of other processes.

Along with these priorities, the block I/O subsystem also allows *weights* from 10 to 1000 to be assigned to control groups. These weights enable finer-grained control over I/O priorities than the sub-priorities supported by the best-effort scheduling class. Control groups can be used to manage virtual machines, containers, and even standard processes all through the same interface.

## IV. Fennel: Architecture and Design

Fennel operates as a user space application, monitoring disk use and environmental conditions to build *scheduling directives* that are passed to the kernel layer through the cgroup interface. Any system scheduler that implements the weighted block I/O resource controller interface can be used in conjunction with Fennel. In this work, we use CFQ as the underlying scheduling mechanism because (1) it implements the necessary weighted I/O interfaces, (2) the algorithm excels at enforcing short-term fairness across equal scheduling weights, and (3) tunable parameters allow us to cope with the particularities of both mechanical and solid state drives. Fennel executes as a system *daemon* and is responsible for:

1) Collecting data about I/O activities for each disk and maintaining a history of requests on a per-process basis
2) Monitoring disk health and utilization
3) Adjusting scheduling priorities with respect to historical trends and usage patterns

These concerns are divided into three components: the process monitor, hardware monitor, and scheduling directive generator.

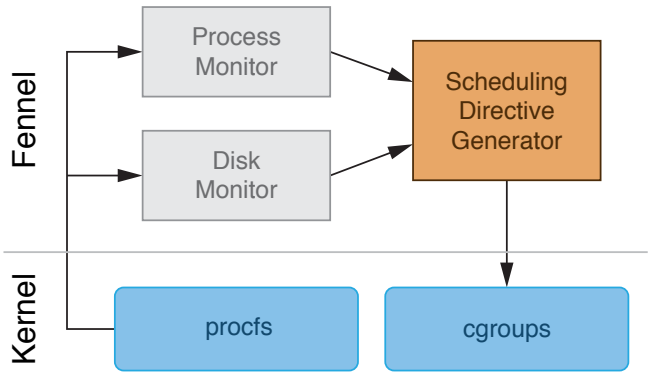


Fig. 2. Interactions between core Fennel components and the kernel. In this illustration, arrows represent the flow of information to and from components: the *procfs* facility is queried to retrieve process and disk hardware information, and then new priorities are published to the scheduler through the *cgroups* interface.

The interactions between components are shown in Figure 2; the activity and hardware monitors collect statistics about disks and processes in parallel, and then the scheduling directive generator acts on the information to update scheduling weights. We designed Fennel as a user space application to ensure portability: communication between the daemon and kernel are abstracted away by a set of interfaces for data gathering and priority modification that can be re-implemented for other operating systems.

### A. Process Monitor

To record information about processes that are performing disk operations, Fennel inspects kernel data structures exposed through the *procfs* interface. A configurable interval (one second by default) determines how often the data structures will be read, and any processes that have performed disk reads or writes will be added to an *active task list* for further analysis. This step also updates historical data about each process running on the system, and removes records for processes that have terminated. When a VM has previously issued I/O requests but has not during the current reporting period, an *idle state* notification is added to its task history data so that the scheduling directive generator knows that the process has been idle. In most configurations, idling will help increase the weight of the VM in question. Collecting this activity information requires iterating through the current set of running processes; on our test cluster, servers that were managing 300-500 processes generated this list in about **81.2 ms** on average (100 iterations per server across 78 servers, with a standard deviation of 7.6 ms).

Each task data structure managed by the process monitor includes a variety of information: a time stamp, the number of bytes read and written during the reporting period, and the amount of time spent waiting in the queue for requests to be serviced. User, process, and thread group IDs are also included to aid analysis by the scheduling directive generator or system administrators. Summary statistics are also maintained, including minimum, maximum, average, and standard deviations for both read and write operations. Once process information has been collected, it is passed to the directive generator. During the collection process, the hardware monitor is also gathering information from disks attached to the system.

### B. Hardware Monitor

Fennel maintains records about each disk at a physical machine to guide its scheduling decisions. This information is crucial in determining the amount of concurrent I/O requests a disk can handle, its average speed and latency between seek operations, and how much a particular access pattern may impact other processes. Table I outlines some of the information gathered by the disk hardware monitor on the drives in our test deployment, which have a range of speed and seek capabilities: large, relatively slow consumer grade storage disks (A, B), small, high-RPM mechanical disks with low platter density but high seek performance (C), and solid

TABLE I
READ, WRITE, AND SEEK PERFORMANCE OF EACH OF THE DISKS IN OUR TEST ENVIRONMENT, AVERAGED OVER 100 ITERATIONS.

| Disk | Model | Manufacturer | Speed | Capacity | Read (MB/s) | Write (MB/s) | Random Seek/s |
|---|---|---|---|---|---|---|---|
| A | ST3000DM001 | Seagate | 7200 RPM | 3 TB | 244.2 | 190.1 | 295.3 |
| B | MB1000GCEEK | HP | 7200 RPM | 1 TB | 139.6 | 109.3 | 341.0 |
| C | DF0300B8053 | HP | 15000 RPM | 300 GB | 178.9 | 141.7 | 544.2 |
| D | MZ-7TE1T0BW | Samsung | SSD (SATA 3) | 1 TB | 438.5 | 386.8 | 21276.0 |

state drives (D) that are much faster and not constrained by physical movements for seeking.

Seek capabilities are particularly important to our algorithm because a process that performs an excessive amount of random seeks on a mechanical disk will result in high utilization but low throughput due to disk head movements. CFQ includes a provision for penalizing *backward seeks* by prioritizing I/O requests that fall in front of the disk head, which reduces the amount of times the head will have to change direction while servicing requests. However, this approach does not penalize processes that seek frequently. For this reason, we impose a *rotational seek penalty* on mechanical disks to discourage access patterns that require a substantial amount of random seeks.

During startup, Fennel determines whether or not each drive is mechanical by inspecting kernel data structures. To handle hot swapping, it also subscribes for callback notifications when new disks are inserted or removed. For mechanical disks, Fennel monitors cumulative read and write throughput as well as disk usage, $u$, which describes the amount of each scheduling quantum the disk controller is busy. The number of processes using the disk, $n$, is also maintained. When at least one process is using the disk, these variables are used to compute the *throughput score* ($ts$):

$$u = \frac{t_{busy}}{t_{quantum}} \qquad ts = \frac{IO_{read} + IO_{write}}{u \times n}$$

If the throughput score for a disk crosses a configurable threshold, the framework will begin inspecting relevant processes to find the culprit(s). The default threshold is set to 20% of the average bandwidth observed on a per-disk basis. For example, if the disk in question can sustain 100 MB/s average transfer speeds under normal operation but Fennel observes an aggregate throughput of 20 MB/s with 100% utilization, the disk will be flagged for further inspection. Note that simply crossing the threshold does not result in priority changes until a definite cause for the abnormal performance is found. This means that a somewhat higher threshold is preferred since false positives at this stage do not result in miss-classification of processes.

Once hardware information has been collected and drives that are experiencing potentially seek-heavy workloads have been flagged, the data structures are passed to the scheduling directive generator to aid in the decision making process. The hardware monitor can also be configured to alert system administrators of possible disk issues that arise over time; for instance, the CRC error rate on a particular disk may be considered normal by the drive manufacturer's SMART parameters, but the Fennel hardware monitor will observe decreased average performance and issue a warning. Thresholds for disk temperatures, queue lengths, and average performance can all be configured to help pinpoint hardware issues in a cluster.

### C. Accounting for Storage Configurations

The type of disks available on a given machine is another important piece of information provided by the Fennel hardware monitor. This includes whether the disk is rotational or solid state, the disk's operational RPMs (HDDs only), sector sizes, as well as alignment and manufacturer information. Solid state drives in particular require special care when using the CFQ scheduler, as its default time slices may be too long to achieve optimal performance. If a solid state drive or enterprise-class RAID is detected by the hardware monitor, Fennel will optionally reconfigure the following CFQ parameters at run time using the *procfs* interface:

- `slice_idle = 0`
- `quantum = 64`
- `group_idle = 1`

Each of these options helps boost the performance of faster hardware configurations. The `slice_idle` parameter controls how long CFQ will wait for requests in order to handle deceptive idleness. Since deceptive idleness is not as critical of an issue on flash media or RAIDs with sophisticated caches, this delay is disabled. By setting `quantum` to 64 from its default value of 8, CFQ will issue more commands to the disk controller in parallel. Finally, the `group_idle` parameter allows CFQ to idle for a short period of time after servicing each logical grouping of threads or processes. This corrects for any losses due to spatial locality that may have been caused by disabling the `slice_idle` parameter. Since most clusters often have a wide range of heterogeneous hardware, automatically configuring these parameters will help ensure high performance with the CFQ scheduler.

### D. Scheduling Directive Generator

Once the process and hardware monitors are finished collecting data, the scheduling directive generator begins its prioritization algorithm. The first step involves analyzing the amount of bytes read and written by each process and then consulting a set of configurable rules to determine new process weights. Both *time* and *throughput* can be used to influence the weights, but Fennel defaults to using disk throughput because it reflects the actual consumption of resources associated with the process. As a process uses more of a resource, its weight is throttled down in configurable increments. We support a linear weight penalty as well as a logarithmic penalty, influenced by the cumulative data accessed by the process $C_T$, a configurable scale factor $S$ (in MB), and $N$, the number of weight classes desired (10 by default):

$$w = max(N - \frac{C_T}{S},\ 1)$$
$$w = max(N - lg(\frac{C_T}{S}),\ 1)$$

Figure 3 illustrates the difference between these two penalty functions. An optional weight of 0 can be enabled to specify the **idle** kernel scheduling class, but this feature is disabled by default to avoid I/O starvation in deployments with several VMs. For the sake of portability, Fennel allows users to configure up to 100 I/O classes, which are mapped to the Linux block I/O priority weights (10 – 1000). Custom penalty curves are also supported for deployments with exceptional use cases. These parameters are used to calculate the final priorities for each process, and the updates are made using the control group interface to the block I/O resource controller.

When a process has temporarily stopped using storage resources without terminating, the scheduling directive generator will remove a configurable amount of cumulative I/O for each *idle state* notification received from the process monitor. This allows processes to regain scheduling weight over time. By default, the idle state weight increase is set to $S/2$ to prevent toggling between weight classes for processes that frequently perform short, intermittent bursts of I/O. As with the other parameters used in Fennel, the *weight increase* ratio is configurable to be more or less aggressive depending on the workload.

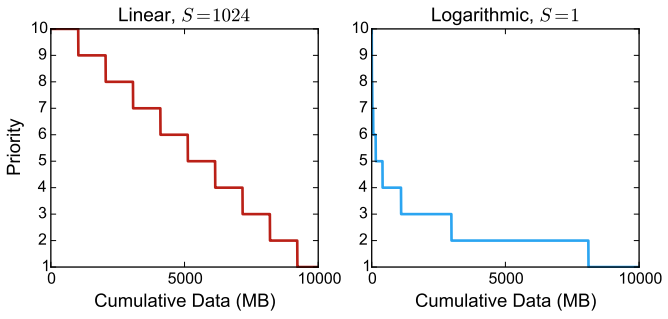


Fig. 3. Configurable weight penalty functions supported by the scheduling directive generator.

After the appropriate weights have been calculated and applied for each process, Fennel inspects the disks that were flagged by the hardware monitor as potential sources of excessive seeks. The daemon determines which processes are accessing the flagged disks using data gathered by the process monitor, and then sorts the processes by cumulative I/O for the current inspection interval; in general, processes that exhibit low cumulative throughput are more likely to be seeking. Fennel then attaches to the processes with the Linux *strace* utility, which allows interactive inspection of all system calls being executed by the process. If a substantial portion of system calls that occur during a one-second sample belong to the `seek()` family of functions and are operating on offsets that do not follow a predominantly sequential pattern, then the process in question is selected for re-prioritization. Note that this secondary step occurs after the first weight adjustment phase to ensure that weight updates are pushed out in a timely manner and not delayed by the excessive seek detection process.

### E. Supported Virtualization Types

While Fennel can be used to manage interference among processes in a cluster, it is also able to alleviate contention for virtual machine workloads. Both Type 1 and Type 2 hypervisors are supported, including KVM virtualization, Xen, and VirtualBox VMs. OS-level virtualization provided through containers can also be managed by Fennel. We designed our framework to be process neutral so it would apply to a broad range of use cases, including situations where there is a mixture of local processes, virtual machines, and containers. In mixed environments Fennel allows certain processes to be selectively blacklisted, which prevents them from being considered by the prioritization algorithm; kernel threads and processes owned by the `root` user are blacklisted in the default configuration.

### F. Commercial and Public Cloud Environments

One of the intended use cases for the Fennel framework is discouraging "noisy neighbors" that consume a disproportionate amount of I/O resources at a single VM, which leads to decreased performance for neighboring VMs on the same host. In fact, Fennel supports *usage tiers* that allow cloud providers to assign priority limits to particular classes of users. If a particular VM is consistently using a large amount of I/O resources, Fennel can also be configured to reassign it to the idle scheduling weight to ensure its disk activities will not affect other users. Ideally, users that are aware of our weighted scheduling scheme would be judicious with I/O operations.

In shared environments, a malicious user could conceivably work around a low scheduling priority by launching a large number of I/O threads. To counteract this behavior, Fennel categorizes thread groups originating from the same user under a single scheduling directive. Virtual machines belonging to the same account are also grouped together to prevent this type of abuse.

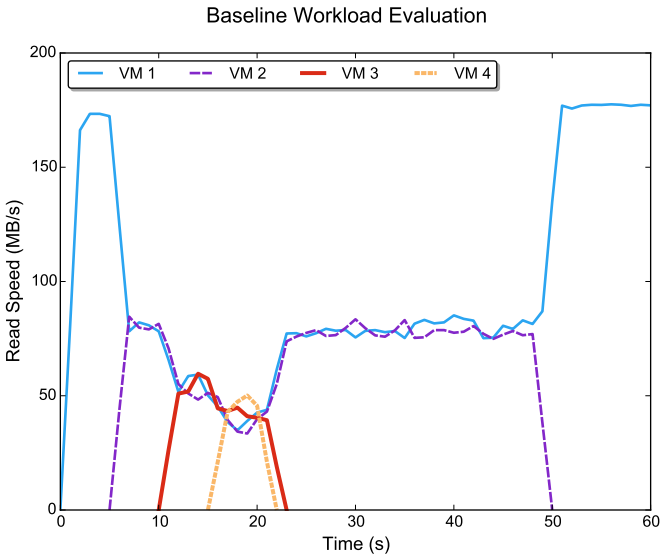


Fig. 4. Read speeds observed across each VM workload on a host without the Fennel daemon. CFQ ensures short-term fairness among the different workloads, which results in longer turnaround times for VM 3 and 4.

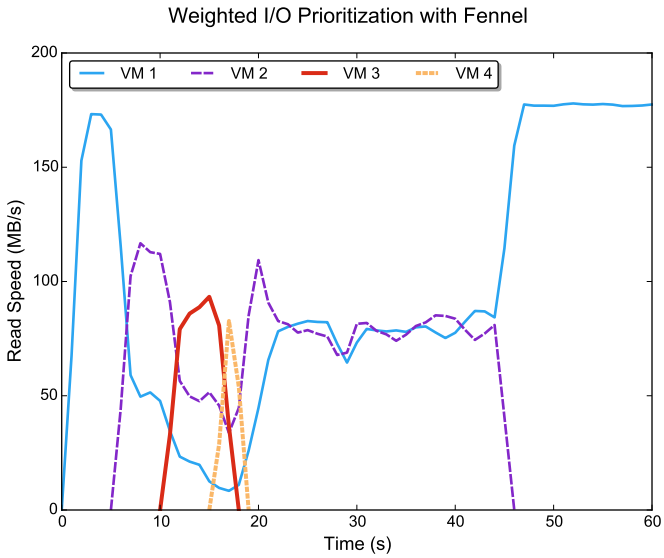


Fig. 5. Read speeds observed across each VM workload with Fennel. Note the visible reduction in completion times for VM 3 and 4, as well as increased bandwidth availability for VM 3.

## V. Performance Evaluation

To evaluate the performance of Fennel, we devised a set of benchmarks for both single-host and distributed settings. These benchmarks include a variety of workloads to help investigate how different usage patterns affect scheduling weights. Our test deployment of 1,200 VMs was populated with a HDFS/Hadoop cluster as well as a Galileo [30], [31] cluster. Galileo is a high-throughput spatiotemporal storage framework that differs from HDFS by using location-aware hashing to distribute incoming data from radars, satellites, and other types of sensors.

### A. Disk Contention

Our first benchmark addresses the performance of Fennel on a single host machine operating four VMs per disk. In this case, we used the following workloads:

1) Hadoop WordCount job processing 10 GB of data from the Common Crawl dataset
2) Hadoop Grep job parsing 3 GB of log files
3) Reading a 500 MB machine learning dataset
4) Converting a 250 MB NetCDF [32] file to the Galileo native storage format

Note that these workloads fall into two categories: short, disk-bound processes (3, 4), and long-running tasks with interleaved processing and I/O (1, 2). Each task was launched sequentially from its own virtual machine with a five-second lag between each launch. The first 60 seconds of this baseline evaluation are shown in Figure 4; CFQ ensures that each process will receive an equal share of the disk, resulting in evenly distributed read speeds throughout the entire benchmark.

In the next benchmark iteration, we activated the Fennel daemon on the machine hosting our four VM configurations. The daemon was configured to use 10 weight classes ranging from the highest to lowest possible priorities and a scale factor of 128 MB. This results in weight changes after each 128 MB increment of cumulative I/O, or, in other words, a process will reach the lowest priority after reading 1,280 MB of data. Figure 5 illustrates the VM performance when operating under the Fennel daemon: while VM 1 is initially de-prioritized heavily due to its higher consumption of I/O resources, it eventually reaches parity with VM 2, a process that is also reading a significant amount of data from the disk.

With Fennel active, VM 3 and 4 receive higher I/O priorities, which translates to faster completion times for their small workloads. It is also worth noting that when VM 1 is the sole remaining task executing on the system, its I/O throughput matches earlier values, indicating that being scheduled at the lowest priority does not incur a throughput penalty when no other processes are accessing the disk. Table II compares completion times between test iterations. VM 3 and 4 clearly benefit from our weighted I/O scheme, finishing 46.4% and 42.7% faster than the baseline test, respectively. Another side effect of Fennel's influence is slight improvements in completion times for the long-running VM instances due to reduced overall contention for resources.

TABLE II
Comparison of completion times for each VM instance.

| VM | Baseline (s) | Fennel (s) | Change (%) |
|---|---|---|---|
| 1 | 83.0 | 81.9 | -1.3 |
| 2 | 44.6 | 40.1 | -10.1 |
| 3 | 12.4 | 7.1 | -42.7 |
| 4 | 6.9 | 3.7 | -46.4 |

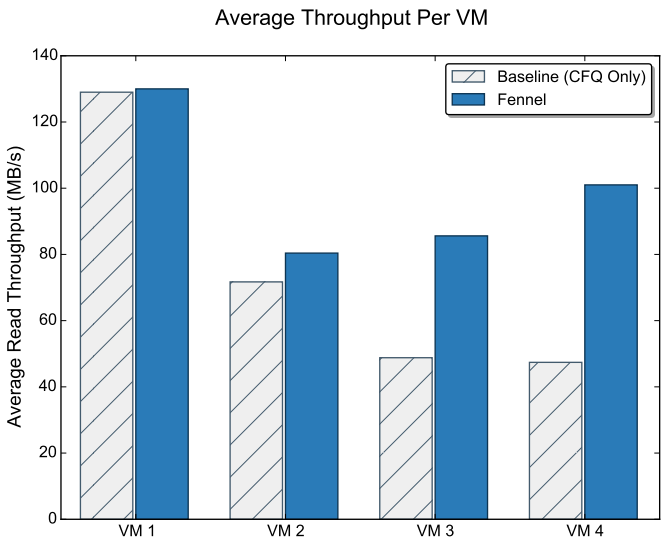


Fig. 6. Throughput observed at each VM in the disk contention test.

In Figure 6, the mean I/O throughput is broken down for each VM in both iterations of the benchmark. As expected, the largest gain in observed throughput is at VM 3 and 4. Additionally, the aggregate throughput for the baseline and Fennel version of the benchmark was 166.0 MB/s and 166.2 MB/s, respectively, demonstrating that Fennel's weighting decisions have not reduced overall throughput.

### B. Large-Scale MapReduce Evaluation

While the previous benchmark demonstrated Fennel's effectiveness at a single node, we also executed a large-scale MapReduce application over the entire 1,200 VM cluster. In this test, a Hadoop Grep job was launched to locate tokens in server log files stored at each VM, with each log containing approximately 1 GB of ASCII data on average. To exercise the Fennel daemons, we also began streaming weather feature data from the western United States (sourced from our NOAA dataset [10]) into Galileo for storage. Since Galileo partitions data spatially, the sample readings from the western US coast selectively impacted storage nodes while leaving others idle.

Figure 7 compares the performance of the MapReduce Grep job running with and without the Fennel daemon activated. Each data point in the figure represents a task completion time, with the longest task determining the overall speed of the job. Note that approximately 200 VMs (900 – 1100) were impacted by the Galileo storage operation, causing interference. In this benchmark, the baseline configuration completed in 45.1 seconds (26.6 GB/s aggregate throughput), while the Fennel configuration took 36.0 seconds to complete (33.3 GB/s aggregate throughput) for 25% improvement in execution time. Additionally, four speculative tasks were launched by the Hadoop runtime during the baseline test, whereas no speculative tasks were launched in the second iteration of the test.

## VI. Conclusions and Future Work

This paper described our framework, Fennel, for accomplishing high-throughput, data-intensive computing in virtualized settings. Fennel is broadly deployable (in physical and virtual machine settings) because its constituent components reside outside the VMs and do not require kernel or hypervisor modifications. Effective prioritization of processes based on the number and type of I/O operations being performed ensures that fairness is achieved without compromising on overall throughput; in fact, our approach ensures that the average time spent by a process waiting to perform I/O is reduced as a proportion of the overall execution times. Data intensive computing often involves data parallel processing – as engendered in MapReduce and frameworks that leverage it – where completion times are determined by the last task to complete. Such tasks are often stragglers and MapReduce runtimes launch speculative, backup tasks for such straggelers. Through its fairness preservation criteria, Fennel alleviates stragglers since tasks are less impacted by interference from collocated noisy neighbors that starve other processes from performing I/O. Our benchmarks were performed in large settings involving thousands of VMs, multiple analytic processes, and voluminous datasets. These experiments demonstrate the effectiveness of ensuring fairness, high-throughput, and faster completion times for data intensive tasks.

As part of our future work we plan to leverage time-series analysis for scheduling I/O processing, dynamic apportioning of virtual memory among collocated virtual machines, and incorporating heat dissipation as a metric to be monitored and reduced in such settings. Another avenue for future research is support for capabilities often required by scientific applications, such as kernel density estimation of attribute values, without having to read all stored data.

## Acknowledgments

This research was supported by funding from the US Department of Homeland Security's Long Range program (HSHQDC-13-C-B0018) and the US National Science Foundation's Computer Systems Research Program (CNS-1253908).

## References

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the Future*, vol. 2007, pp. 1–16, 2012.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, 2008.

[3] The Apache Software Foundation. Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[6] The Apache Software Foundation. Apache Pig. [Online]. Available: https://pig.apache.org

[7] The Linux Foundation. 2014 Enterprise End User Report. [Online]. Available: http://www.linuxfoundation.org/publications/linux-foundation/linux-end-user-trends-report-2014

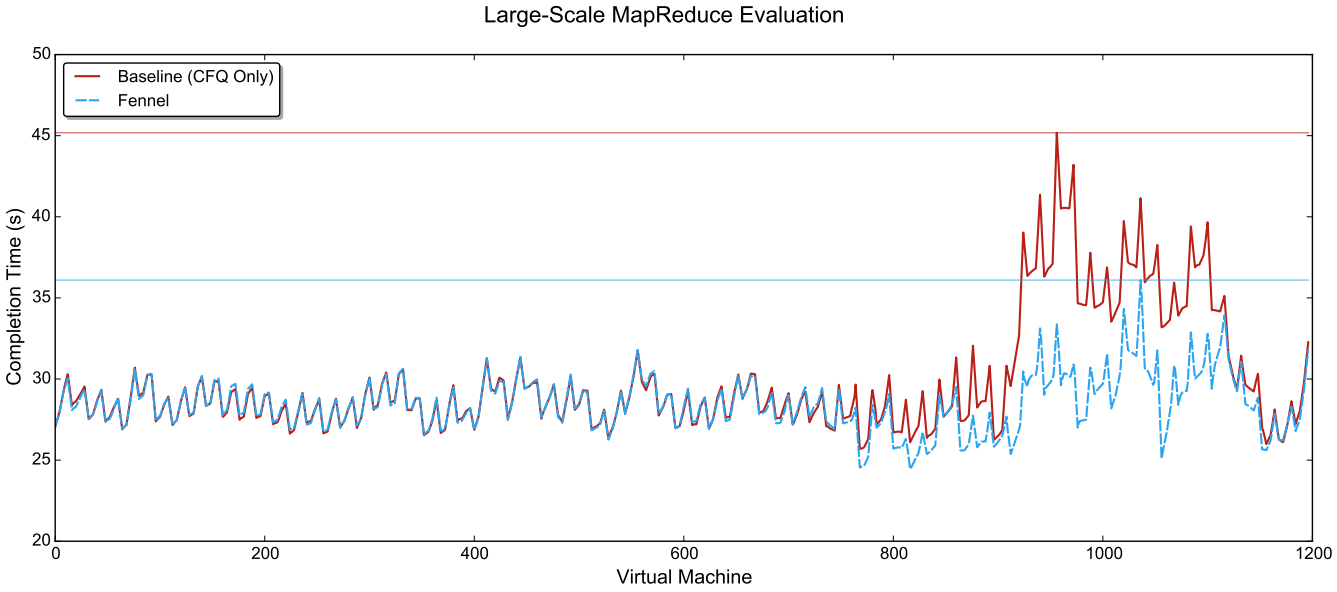


Fig. 7. Hadoop task completion times for the entire 1,200-VM cluster while experiencing interference across a subset of the VMs. In this benchmark, the baseline configuration completed in 45.1 seconds, whereas the test iteration with Fennel enabled completed in 36.0 seconds.


[8] J. Axboe, "Linux block IO – present and future," in *Ottawa Linux Symp*. Citeseer, 2004, pp. 51–61.

[9] The Common Crawl Foundation. (2015) Common Crawl Corpus. [Online]. Available: http://commoncrawl.org/

[10] National Oceanic and Atmospheric Administration. (2015) The north american mesoscale forecast system. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM

[11] D. M. Jacobson and J. Wilkes, *Disk scheduling algorithms based on rotational position*. Citeseer, 1991.

[12] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 117–130, Oct. 2001. [Online]. Available: http://doi.acm.org/10.1145/502059.502046

[13] Q. Zhang, D. Feng, F. Wang, and Y. Xie, "An efficient, qos-aware i/o scheduler for solid state drive," in *High Performance Computing and Communications 2013 IEEE International Conference on*, Nov 2013, pp. 1408–1415.

[14] M. Bjørling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block io: Introducing multi-queue ssd access on multi-core systems," in *Proceedings of the 6th International Systems and Storage Conference*, ser. SYSTOR '13. New York, NY, USA: ACM, 2013, pp. 22:1–22:10. [Online]. Available: http://doi.acm.org/10.1145/2485732.2485740

[15] S. Park and K. Shen, "FIOS: A fair, efficient flash i/o scheduler," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208461.2208474

[16] 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*, Aug 2008, pp. 647–653.

[17] Y. Wu, B. Jia, and Z. Qi, "Io qos: A new disk i/o scheduler module with qos guarantee for cloud platform," in *Information Science and Engineering (ISISE), 2012 International Symposium on*, Dec 2012, pp. 441–444.

[18] J. Turner, "New directions in communications(or which way to the information age?)," *IEEE communications Magazine*, vol. 24, no. 10, pp. 8–15, 1986.

[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 3, pp. 344–357, 1993.

[20] 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*, ser. VEE '07. New York, NY, USA: ACM, 2007, pp. 105–115. [Online]. Available: http://doi.acm.org/10.1145/1254810.1254826

[21] D.-J. Kang, S.-I. Jung, R. Tsuruta, and H. Takahashi, "Range-bw: I/o scheduler for predicable disk i/o bandwidth," in *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*, vol. 1, March 2010, pp. 175–180.

[22] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passe?" *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 20–24, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1740390.1740396

[23] IBM Corporation, "Best practices for KVM — best practices for block I/O performance," 2012.

[24] Red Hat, Inc. (2015) What is the recommended I/O scheduler for a database workload in Red Hat Enterprise Linux? [Online]. Available: https://access.redhat.com/solutions/54164

[25] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drives," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 295–304. [Online]. Available: http://doi.acm.org/10.1145/1629335.1629375

[26] P. McKenney, "Stochastic fairness queueing," in *INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE*, Jun 1990, pp. 733–740 vol.2.

[27] R. Rosen, "Resource management: Linux kernel namespaces and cgroups," *Haifux, May*, 2013.

[28] ——, "Linux containers and the future cloud," *Linux Journal*, vol. 2014, no. 240, Apr. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2618216.2618219

[29] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[30] 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.

[31] ——, "Expressive query support for multidimensional data in distributed hash tables," in *Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, nov. 2012.

[32] R. Rew and G. Davis, "Netcdf: an interface for scientific data access," *Computer Graphics and Applications, IEEE*, 1990.