Responding Rapidly to Service Level Violations Using Virtual Appliances

Lakshmi N. Bairavasundaram
NetApp, Inc.

Gokul Soundararajan
NetApp, Inc.

Vipul Mathur
NetApp, Inc.

Kaladhar Voruganti
NetApp, Inc.

Kiran Srinivasan
NetApp, Inc.

Abstract

One of the key goals in the data center today is providing storage services with service-level objectives (SLOs) for performance metrics such as latency and throughput. Meeting such SLOs is challenging due to the dynamism observed in these environments. In this position paper, we propose dynamic instantiation of virtual appliances, that is, virtual machines with storage functionality, as a mechanism to meet storage SLOs efficiently.

In order for dynamic instantiation to be realistic for rapidly-changing environments, it should be automated. Therefore, an important goal of this paper is to show that such automation is feasible. We do so through a caching case study. Specifically, we build the automation framework for dynamically instantiating virtual caching appliances. This framework identifies sets of interfering workloads that can benefit from caching, determines the cache-size requirements of workloads, non-disruptively migrates the application to use the cache, and warms the cache to quickly return to acceptable service levels. We show through an experiment that this approach addresses SLO violations while using resources efficiently.

1. INTRODUCTION

Cloud environments are highly dynamic: workload requirements change significantly over time [8, 11, 25], and workloads (often in the form of virtual machines) are created and removed at a fast rate [6]. This dynamism greatly impacts the goal of providing service-level objectives (SLOs) such average latency or target bandwidth for access to storage. Administrators either provision storage resources for the peak workload and waste resources, or they provision aggressively and are unable to act quickly enough to handle workload changes.

Various techniques [3, 12, 13, 14, 17, 33] have been proposed for handling scenarios where SLOs are violated due to workload dynamism. These techniques primarily use resources on storage systems to handle SLO violations [12, 33], limiting their applicability. Further, they either provide very temporary relief at the expense of lower priority workloads [13, 33] or they require a long time to become effective [3, 12, 17].

We develop a new SLO violation-handling technique that addresses the limitations of current mechanisms. We propose that virtual appliances, that is, storage functionality in virtual machines, be created dynamically to handle violations appropriately. A variety of storage-related virtual appliances can be created dynamically. Examples include the creation of a workload-characterization appliance when a workload is under consideration for migration (or other changes), and the creation of a compression appliance when the storage system is about to run out of space. These activities are high-overhead and temporary, and allocating resources permanently may be expensive. Many virtual appliances are available commercially, including, NetApp® Data ONTAP® Edge [24], Amazon ElastiCache [2], and VMware vSphere® Storage Appliance [32].

In order to illustrate dynamic instantiation of virtual appliances, we consider a case study that uses virtual caching appliances (VCAs) to absorb I/O load during workload peaks. The approach provides a violation-handling mechanism for peaks that occur up to a few times a day and last a few hours, which is a common occurrence [11]. This approach helps storage systems meet SLOs by using compute-server resources in addition to storage-system resources.

We have built an automation framework called *Dynamite* for dynamic instantiation of VCAs. In particular, we use FlexCache® [23] technology based on Data ONTAP Edge [24] as the VCA. Dynamite performs a series of steps to enable the automation, including detection of interference between workloads, estimating the cache-size requirements of each workload, creating the VCA and re-routing application traffic, and warming up the cache to restore SLO conformance quickly. We conduct an experiment with interfering workloads to show that our Dynamite prototype dynamically instantiates a VCA to handle an SLO violation quickly.

The rest of the paper is structured as follows. We present detailed motivation for our work in Section 2 and discuss related work in Section 3. We describe dynamic instantiation in Section 4. We then detail the Dynamite workflow in Section 5. Finally, we present our prototype in Section 6, our evaluation in Section 7, and conclusions in Section 8.

2. MOTIVATION

We provide a background on the three trends in data centers today – dynamism, sharing, and scale – and describe why an automated approach is needed to address these trends.

2.1 Dynamism

There are two sources of workload dynamism in data centers. The first source is the change in a workload's requirements over time [8, 11, 25]. Specifically, peak workload requirements can be much higher than the average requirements. Figure 1 shows the variation in IOPS (measured hourly) over time for two workloads from a set of traces from MSR Cambridge [22]. The peak IOPS is 6 times the average IOPS for the first trace (USR-1) and is 15.5 times the average IOPS for the second trace (SRC1-1). In fact, in more than 20 of the 36 traces from MSR [22] that we have examined, the peak is at least 10 times the average. We also note that in SRC1-1, the occurrence of peaks is periodic in nature (first hour of each day). The second source of dynamism in the data center is addition and removal of workloads over time, contributing to the changes in overall workload observed by storage systems. Such dynamism is especially true for cloud service providers, where many virtual machines are created or destroyed at a fast rate (*e.g.*, EC2 instances [6, 7]).

Dynamism causes one of two problems: administrators either provision resources for the peak workload thereby wasting resources, or they provision aggressively and are unable to act quickly enough to handle changes in requirements, thereby violating SLOs. In order to enable aggressive provisioning and efficient resource utilization, we need techniques to quickly mitigate SLO violations. Current mitigation techniques are either temporary fixes (*e.g.*, throttling workloads [10, 18, 13, 35]) or operate over very large timescales (*e.g.*, data migration [1, 3, 4]). As shown in Figure 1 and discussed elsewhere [11], workloads may experience peaks a few times a day. Thus, techniques that become effective in less than an hour are particularly useful.

2.2 Sharing

Multiple applications now share storage resources. While this approach increases resource utilization, sharing implies that the performance of one workload can be affected by the activities of another with which it shares storage resources. Administrators often have little visibility into exactly how resources are shared and what workloads interfere, thereby forcing more conservative provisioning practices. Even so, errors in performance modeling (both human and automated) for multiple workloads can cause SLO violations that are hard to diagnose or handle.

2.3 Scale

Another obvious trend is the large scale of operations in the cloud. While the large scale is a core element in the business model for the cloud, managing workloads at scale is particularly hard. Human intervention in handling every SLO violation (or resource crunch) is prohibitively expensive if not impossible. Therefore, automation of management activities is essential; the research on management using service-level objectives [1, 14, 13, 21, 36] is particularly relevant and we leverage (and assume the availability of) such techniques in this work. We expect that an application's SLO for performance is specified in storage terms such as access latency or IOPS, or can be translated to storage terms.

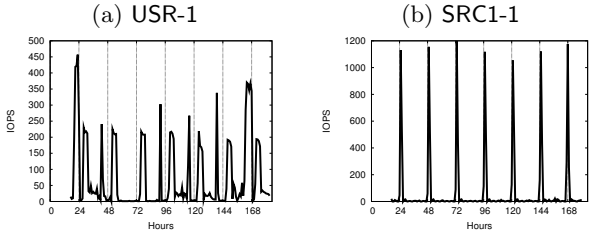


Figure 1: **Workload Dynamism:** *The figures show the IOPS (measured hourly) issued by the MSR Cambridge traces* USR-1 *and* SRC1-1. *The traces start at the 17-hour mark. The grid lines represent the last hour of each day.*

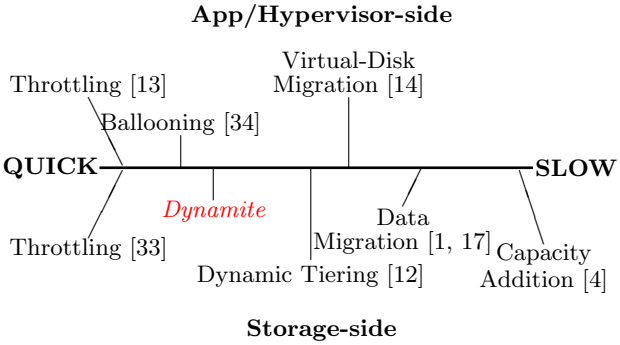


Figure 2: **Mitigating SLO Violations:** *We show the timescale (a few seconds to many months) and location (client-side vs. storage-side) at which techniques for mitigating SLO violations operate.*

3. RELATED WORK

We discuss related work from the viewpoint of SLO-based management for storage systems, cache sizing [16, 20, 30, 33, 34, 37] and cache warmup [15].

SLO-based Management: A variety of research efforts have contributed greatly to the field of SLO-based management, especially for storage systems. These efforts have created a workflow for automated management, including techniques for monitoring [29], analysis [1], planning [1, 3], and execution of various mitigation techniques [12, 13, 14, 33, 34]. Figure 2 shows a wide range of SLO-violation correction techniques. The figure classifies the various techniques based on: (1) the amount of time it takes to initiate and complete a particular mitigation technique, and (2) location (client/storage) where the actions are performed.

We now briefly describe work on each of the techniques. Workload-throttling controls the entry of I/O requests based on the workload's priority or its share of resources [13, 33]. These techniques handle violations for a short period and longer-term solutions are required for persistent violations. Ballooning techniques have been proposed for VMs to dynamically acquire more memory resources [34]. Such techniques are effective as long as the requirements of the VM can be satisfied using the resources on the same physical machine. Schemes to partition available caches based on requirements have been examined [33]; Dynamic load-balancing

across storage nodes [14, 17], or between flash and disk tiers [12] have also been proposed to mitigate SLO violations. Mechanisms that allow for adding more capacity by monitoring a system, and subsequently proposing a new configuration have also been proposed [1, 3, 4]; Finally, in terms of dynamic instantiation, a variety of automated scaling approaches have been suggested both for computation [25] and for distributed storage nodes [17].

Cache Sizing and Warmup: Methods to determine the cacheability of workloads include using new hardware [28, 37], to using OS memory management features [34, 37], to running trace-based simulations [16], and to working-set estimations [30, 34]. The focus on cache warmup has primarily been from a benchmarking perspective, where techniques for accelerating simulations by reducing warmup periods have been explored [15].

Our automation framework (Dynamite) is the *complete management workflow* needed for dynamic instantiation of VCAs. Our approach is complementary to the above techniques. It operates on a timescale of a few hours to days, to a limited extent similar to that of ballooning [34], cache partitioning [33], and dynamic tiering [12]. Our technique differs from these approaches primarily in acquiring non-local resources, and in efficiently and accurately estimating resource needs.

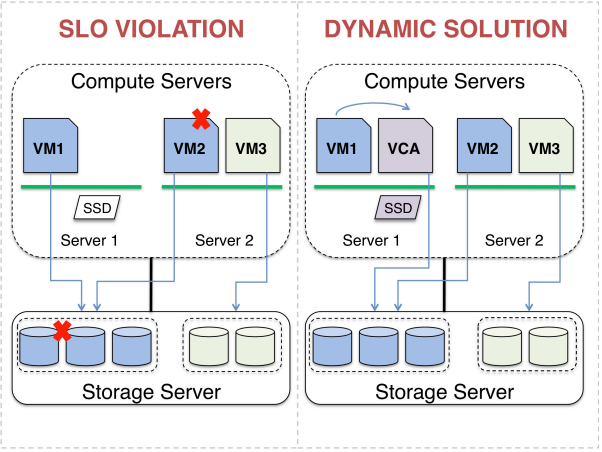


Figure 3: **Dynamism:** *SLO violations can be handled by dynamically creating VCAs to offload read IOs from the storage system.*

4. DYNAMISM AS AN OPPORTUNITY

We propose that functionality in the form of virtual machines be dynamically inserted into the storage stack when needed. We refer to such functionality as *virtual appliances*. We acquire or discard virtualized compute resources based on demand, thereby reducing resource overprovisioning.

While the dynamic-instantiation approach implies "shifting" of resource requirements (and hence the hardware purchased) from storage systems to compute servers, such a shift can be particularly beneficial. Resources on compute servers are typically cheaper than those on storage systems due to commoditization. Further, resources on compute servers can be used flexibly; *e.g.*, DRAM on compute servers can be used for compute needs of applications as well as for caching needs of the storage stack. A number of storage-related virtual appliances can be created dynamically. One can create a workload-characterization appliance interposed between the client and storage server when a workload is under consideration for migration (or other changes). One can also create a compression/deduplication appliance that can be inserted when the storage system is about to run of capacity. These activities are high-overhead and temporary, and allocating resources permanently may be expensive.

In this paper, we explore the creation of a *virtual caching appliance (VCA)* on demand for handling workload peaks when storage resources are provisioned for the average workload. Figure 3 shows a scenario where a VCA is dynamically instantiated to handle an SLO violation. It shows two compute servers containing three VMs (VM_1, VM_2, VM_3) using a network-attached storage server; VM_1 and VM_2 share a set of disks while VM_3 uses a different set. Let us assume that due to a change in workload, VM_2 is not meeting its average latency SLO. We can create a VCA for VM_1 (which interferes with VM_2) and reduce the load on the storage system, thereby enabling VM_2 to meet its SLO.

Most of the *mechanisms* for delivering storage or caching functionality in the form of virtual appliances have already been developed [2, 32]. The ability to instantiate them on demand (*e.g.*, Amazon ElastiCache [2]) has been developed as well. Therefore, our focus is on techniques needed to address *policy* questions; examples include which workload should be cached, how much cache space to allocate (*cache sizing*), and what data should be used to warm up the cache (*cache warmup*). Currently, activities related to policy questions are performed manually (*e.g.*, manual cache sizing [2]). Human intervention in answering policy questions or in dynamically instantiating VCAs is near-impossible due to the rapid changes in workload characteristics, the high degree of resource sharing across workloads, and the large scale of data-center operations. Our Dynamite workflow aims to automate such policy decisions.

5. DYNAMITE WORKFLOW

We describe the Dynamite workflow for dynamically instantiating VCAs to meet performance SLOs. Figure 4 shows the Dynamite workflow. We organize the steps taken by Dynamite into three phases: *monitoring*, *analysis and planning*, and *execution*. We explain each of these phases in detail below, using the scenario in Figure 3 as an example. We also highlight each numbered step shown in Figure 4 in bold.

5.1 Monitoring

Dynamite utilizes the performance and traffic metrics of each workload running on the storage system including I/Os per second, average latency, request sizes, and operation mix. In addition, Dynamite tracks characteristics such as working-set sizes over time and most-frequently-used zones of data. These two metrics are very useful for answering policy questions.

(Step 1) SLO Violation Detection: The performance metrics are used to detect situations when a workload is not meeting its SLO. Figure 3 shows that VM_2 is in violation; this detection triggers the rest of the Dynamite workflow to correct the problem.

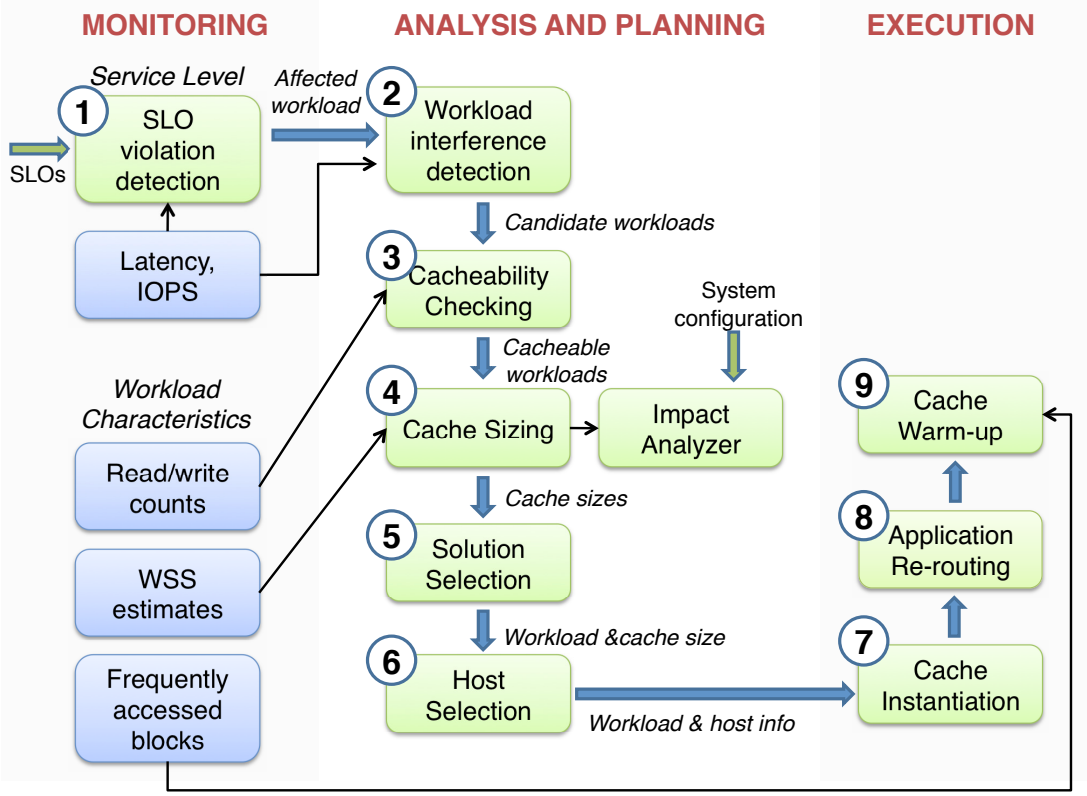


Figure 4: **Dynamite Workflow:** *We organize the steps taken by Dynamite into three phases: monitoring, analysis and planning, and execution. The monitoring phase continually tracks workload characteristics and an application's compliance with respect to its SLOs. Upon detection of a SLO violation, the analysis and planning phase looks at interfering workloads, selects a workload to cache, and sizes the cache appropriately. Once the solution is determined, a new cache is instantiated and the workload is re-routed to use the cache.*

5.2 Analysis and Planning

Upon the detection of an SLO violation, Dynamite identifies interfering workloads, finds cacheable workloads, builds a list of caching solutions, chooses the best solution, and finds a compute server to instantiate the VCA.

(Step 2) Workload Interference Detection: Dynamite identifies a set of interfering workloads, *i.e.*, workloads that compete for the same resources as the *affected workload* (workload experiencing an SLO violation), on the storage system. The insight is that offloading the affected workload or any of the interfering workloads would reduce resource contention; by identifying interfering workloads, Dynamite has more options for workloads to offload than just the affected workload or any peaking workload. In Figure 3, VM_1, VM_2, and VM_3 could all interfere with each other. VM_2 and VM_3 could interfere if there is a memory or network bottleneck; and VM_1 and VM_2 could interfere due to a disk bottleneck as well. Dynamite uses statistical correlation between changes in performance metrics of different workloads to determine whether they are interfering[1]. In our example, let us assume VM_1 and VM_2 are identified as interfering.

(Step 3) Cacheability Checking: Next, Dynamite identifies the subset of *cacheable* workloads, that is, workloads that can be offloaded using a cache. First, Dynamite filters workloads based on a set of simple rules; *e.g.*, if the VCA is a write-through cache, workloads with a high fraction of writes cannot be offloaded. The rules can be specified by the VCA vendor as a best-practice. In Figure 3, let us assume that VM_2 has been removed from further consideration as it fails one of the conditions of cacheability checking.

(Steps 4, 5) Cache Sizing and Solution Selection: For the cacheable workloads, Dynamite sizes the VCAs appropriately such that a sufficient fraction of I/Os are offloaded

[1]Alternately, Dynamite can leverage "white-box" knowledge that workloads share disk drives to detect interference candidates.

to the VCA to correct the SLO violation. In particular, Dynamite uses an online cache-sizing tool that uses working-set size estimates gathered by the storage system to build a *miss-ratio curve* (the details of this technique are beyond the scope of this paper). The miss-ratio curve indicates the fraction of I/Os sent to the underlying storage system for various cache sizes. In our example, Dynamite may determine that VM_1 will have a miss ratio of 50% for a 4GB cache, 25% for an 8GB cache, and 20% for a 16GB cache. Next, Dynamite uses a simple performance model to analyze the impact of the cache-miss workload on the storage system and interfering workloads; more advanced impact-analysis techniques [1] can be used if available. Based on the analysis, Dynamite generates a list of (workload, cache-size) solutions that can fix the SLO violation; in the example, it may pick an 8GB cache for VM_1 to offload 75% of the I/Os. This list of solutions is ranked by the amount of resources required for different workloads and the solution with the least resource cost is selected. Additional heuristics such as a preference for non-peaking workloads or workloads with stable working sets may also be used to rank solutions.

(Step 6) Host Selection: The location of each application VM is provided to Dynamite at workload creation time. Dynamite selects the compute server for the VCA using location-based affinity; that is, Dynamite selects the application VM's compute server as long as sufficient resources are available. This approach, and extensions to compute servers on the same rack [8], enables the VCA to be more frugal in using network bandwidth. When sufficient resources are not available locally, Dynamite uses the first server in its search that has sufficient resources. Thus, in Figure 3, the VCA is created on the same compute server as VM_1. One may leave host selection to hypervisor-management tools as long as inter-VM affinity can be specified.

5.3 Execution

The execution steps instantiate and warm up the cache.

(Steps 7, 8) Cache Instantiation and Re-routing: Dynamite invokes hypervisor APIs to create and configure the VCA. Once the VCA has started, the workload to the storage system is re-routed to use the VCA instead. Table 1 presents a variety of non-disruptive options for such re-routing. These options are also non-invasive from the application viewpoint (*i.e.*, no change to the client). Of these options, we use IP-address migration for Dynamite. We first enable dynamic re-routing by allocating a new IP address for data access from the storage system at application VM *creation* time. Dynamic re-routing is performed by migrating this IP address to the VCA. Thus, the approach treats IP addresses as an additional resource that may be consumed to enable re-routing.

(Step 9) Cache Warmup: A cache is not effective unless it is warm. The simplest approach is to allow the cache to be warmed by application access but it may not restore the system to SLO-conformance quickly enough. Therefore, Dynamite performs explicit cache warmup. Explicit cache warmup may generate more load on a heavily-loaded storage system, potentially causing more SLO violations. At the same time, it may allow violations to be of shorter duration than without warmup; further, warmup may be particularly useful when VCA instantiation is performed proactively before violations occur. Dynamite monitors the working set at a large granularity (called *zones*) and reads entire zones sequentially. In particular, Dynamite monitors only the heavily-accessed zones (more than 0.1% of accesses) using the lossy-counting algorithm [19].

Cache Removal: The workloads, storage system, and VCA continue to be monitored once the VCA is created. The cache can be removed when: (1) the workload reaches a sustained low-load phase, (2) the resource utilization levels on the storage system drop, or (3) the workload changes so that the cache becomes ineffective. Thus, resources are returned to the compute pool when they are no longer needed. Application traffic is also re-routed back to the storage system prior to cache removal. Given that the decision-making process for removal uses the components of the workflow above, we do not focus on cache removal in the rest of the paper.

6. DYNAMITE PROTOTYPE

Our implementation of Dynamite is split between a management server and a trace-replay tool. The management server collects configuration information and periodic (every 10s) metrics from the storage system and compute servers using the appropriate APIs. This configuration information includes the available CPUs, DRAM, and storage space. The metrics collected include IOPS, the average latencies of I/Os, and the bytes transferred over the network interfaces. The storage-system metrics are used to detect SLO violations and interference between workloads while the compute-server metrics are used to determine available capacity for the VCA. Upon the detection of an SLO violation, the rest of the Dynamite workflow is triggered. For ease of implementation, we use a vanilla storage system and implement the working-set estimation and tracking of heavily accessed zones within the tool used to replay our traces; given their low overhead, we expect these monitoring features to be implemented in future storage systems.

VCA: The Dynamite prototype uses NetApp FlexCache [23] technology packaged as a virtual machine as the VCA. FlexCache supports the Network File System (NFS) Version 3 protocol [9] for client data access. The absence of server state (sessions, etc.) in NFSv3 allows non-disruptive traffic re-routing through IP-address migration. Even so, Dynamite configures the VCA to ensure that NFS file handles are generated with the same inputs and processed in an *identical* fashion to the storage server. This configuration ensures that operations to open files are processed correctly.

FlexCache uses a proprietary protocol to perform reads and writes to the origin storage system. It forwards all writes to the storage system and invalidates the data blocks that are written to. The protocol also provides cache coherence enabling multiple FlexCache instances to be instantiated for the same data. FlexCache uses both memory and disk (or SSDs) to perform caching; the data in memory is a subset of the data on disk. Dynamite configures the VCA with four virtual CPUs, the amount of memory determined by cache sizing, and selects local drives for storing its virtual disks. In our experiments, we use SSDs available on compute servers. FlexCache supports a read command that causes no data transfer to the client, but causes the data to be read internally from the backend storage system. Dynamite uses this command for warming the cache.

While we have used the FlexCache VCA for our prototype, we believe that Dynamite can be readily applied to other VCAs (such as Amazon ElastiCache [2]).

Technique	Description	Protocol	Constraint
IP-address migration	Use unique IP addresses for storage access for each workload; Move address to VCA	NFSv3	IP addresses
pNFS layout revoke [26]	pNFS allows storage metadata servers to revoke data layouts and trigger the client to re-fetch the layout, at which time the server can direct the client to the VCA.	NFSv4.1	None
Null Copy-offload	Trigger storage migration [31] to the VCA. Hypervisor offloads the copy (*e.g.*, SCSI Extended Copy [27]) to the storage, which does nothing and returns success.	SCSI/ NFS	Hypervisor dependent
Permanent proxy	All applications access storage through a proxy that can perform load-balancing, cache routing, etc. (inefficient use of resources since it is "always-on")	Proxy dependent	Resources used by proxy

Table 1: **Traffic Rerouting:** *Set of options for rerouting application traffic non-disruptively, where disruption is defined as a long (many minutes) stoppage of the application. We use IP-address migration for Dynamite.*

7. EVALUATION

We present a case study showing how a VCA can be instantiated to offload IOs from the storage system. We focus on showing the benefit of the Dynamite workflow, highlighting the interference detection, cache sizing, and cache warmup aspects. Other parts of the Dynamite workflow are not discussed in detail.

We consider two usage scenarios: reactive deployment and proactive deployment. With reactive deployment, the Dynamite workflow starts only when an SLO violation is detected. Therefore, Dynamite uses recent data on working-set sizes, heavily-accessed zones, etc. for decision making. With a proactive deployment, Dynamite operates in an environment with prior knowledge of workload peaks [11]) and the workload characteristics during peaks. Therefore, Dynamite creates the VCA well before the workload peak actually occurs. We additionally simulate this mode by having Dynamite use working-set size and zones data generated through the workflow for accesses during the workload peaks that are about to occur (instead of using past data).

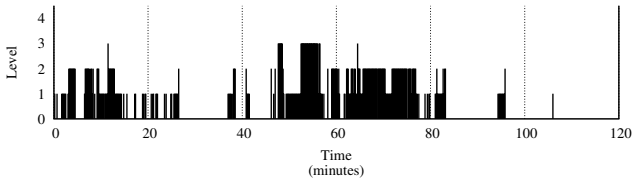
7.1 Experimental Setup

Table 2 shows the configuration for systems used in the experiment. The monitoring server and the Dynamite workflow run on a separate management VM. The trace-replay tool generates I/O requests to the storage system. The tool issues all I/Os asynchronously; that is, it follows timestamps for every I/O on the trace irrespective of whether previous I/Os have been completed. The parallelism in I/Os is limited by the buffer size for outstanding I/Os in Linux® asynchronous IO (`libaio`) – set to 16 I/Os in our setup.

We evaluate Dynamite by replaying the USR-1 trace from a set of MSR traces [22] and a trace generated using the micro-benchmark FIO [5] performing random reads. These workloads share a 11-disk RAID-4 group. We set a simple performance SLO in terms of IOPS where we require 1800 IOPS for USR-1 and 5000 IOPS for FIO; *low-load* situations are not flagged as SLO violations. Using these SLOs, we provision the storage system to meet the average demands of both applications but not the peak demands of both. Each experiment consists of running an intense 2-hour segment of the traces using our trace-replay tool; Even so, we replay USR-1 – one of the more intense MSR traces – at 10x speed since the trace did not stress the storage system at normal speed. Finally, the storage system has a (minor) background workload for administrative tasks (SYS) running concurrently with the workloads.

W	USR1	FIO	SYS	Set#
USR1	1	1	0	1
FIO	1	1	0	1
SYS	0	0	1	2

(a) Interference table



(b) Interference between USR-1 and FIO over time

Figure 6: **Workload Interference:** *We show (a) the intereference table created by Dynamite when SLOs are violated and (b) the intereference measured by Dynamite between the workloads* USR-1 *and* FIO. *Note that higher interference is recorded when the workload SLOs are affected in Figure 5.*

7.2 Results

Figure 5 shows the performance of both workloads where x-axis shows the time elapsed in the experiment and the y-axis shows the IOPS observed by the trace-replay tool. Reactive deployment is shown in Figure 5(a) and proactive deployment is shown in Figure 5(b). The *Alone* and *Co-located* lines are identical for both reactive and proactive.

We first run each workload alone. The workload USR-1 has four distinct phases: a short duration load spike (up to the 7-minute mark), a calm period (between 7 and 48 minutes), a sustained peak load (between 48 and 75 minutes), and a final calm period. FIO performs I/Os at a steady rate of 5000 IOPS. The storage system is able to meet the 1800 IOPS and 5000 IOPS SLOs when the workloads are run in isolation.

Next, we run USR-1 and FIO at the same time, depicted as *Co-located* on the graphs. As the storage system is provisioned for average load, the performance of FIO suffers during phase 1 (the initial load spike) as well as phase 4 (the sustained peak load); in these phases, the average IOPS for FIO is only 3000 (approx.). Similarly, USR-1 does not meet its SLO of 1800 IOPS and the backlog causes the peak section

	Storage Server	Compute Server	App VMs
OS	Data ONTAP	VMware® ESX® 4.1	Ubuntu 10.04
CPU	4 AMD Opteron	8 Intel® Xeon® E5630	1 Logical
RAM	16GB	24GB	512MB
Storage	11 15k RPM FC Disks	3 Samsung MZ-5PA256 SSDs	NFS datastore
Network	1Gbps	1Gbps	1Gbps

Table 2: **Platform Details:** *Details of the hardware and software running on our storage and compute servers.*

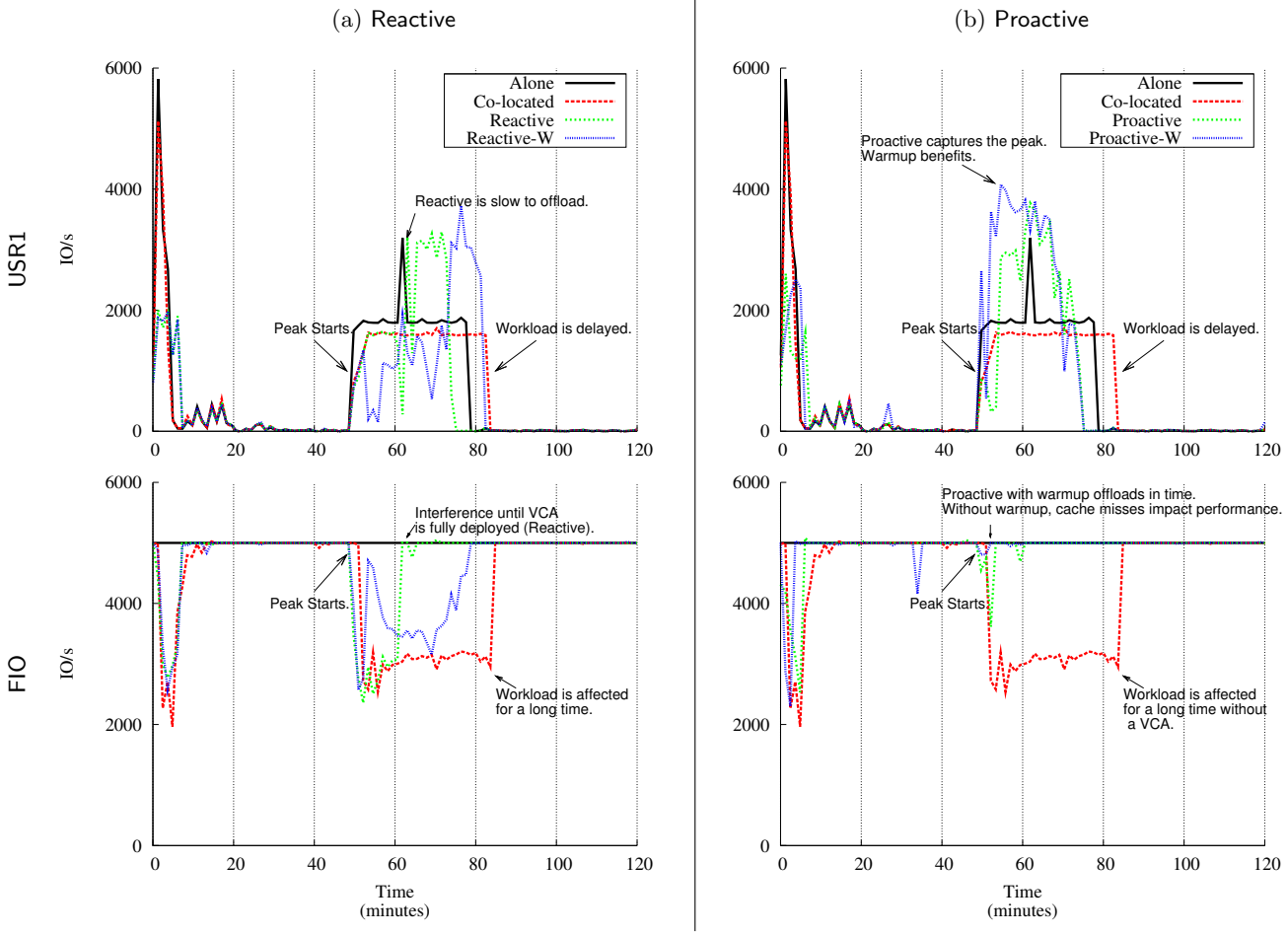


Figure 5: **Results:** *We run a case study with two workloads:* USR-1 *from MSR and a micro-benchmark* FIO *sharing a set of disks on the storage system. The storage system is provisioned to meet the peak load of each workload running alone but not together. We run Dynamite using proactive and reactive VCA instantiation. Our results show that Dynamite correctly sizes the cache, and creates the VCA. Both proactive and reactive schemes address the SLO violation; the proactive, by creating the VCA ahead of time, has no violations during the peak load.*

of the trace to be completed roughly 5 minutes later than it did when the workload was running in isolation. When Dynamite is used in this scenario, the SLO violation is handled efficiently. We now describe the results for each step of Dynamite, discussing reactive deployment first.

Reactive Deployment: Reactive deployment is depicted by lines *Reactive* and *Reactive-W* (reactive with warmup) on the graph. Dynamite ignores the short-duration load spike; these are better handled through techniques such as workload throttling (see Figure 2). Dynamite then detects the occurrence of an SLO violation at the 48-minute mark and performs the following main steps: interference detection, cache sizing, cache instantiation, and cache warmup.

Figure 6(a) shows the interference table generated by Dynamite when the violation is detected. Dynamite detects that workloads USR-1 and FIO are the interfering workloads, and that SYS does not impact the other workloads. Since workload-interference detection is performed constantly, we show in Figure 6(b) the interference levels detected between USR-1 and FIO over the entire experiment for the *Co-located* case. We see that a higher level of interference is correctly detected when USR-1 peaks.

Once Dynamite detects interfering workloads, it determines the cache requirements for the workloads. Figure 7 shows the simulated (using LRU simulator) and estimated (mechanism not described) miss-ratio curves for both workloads. Since these curves apply to reactive deployment, they are calculated using the WSS data collected up to the peak

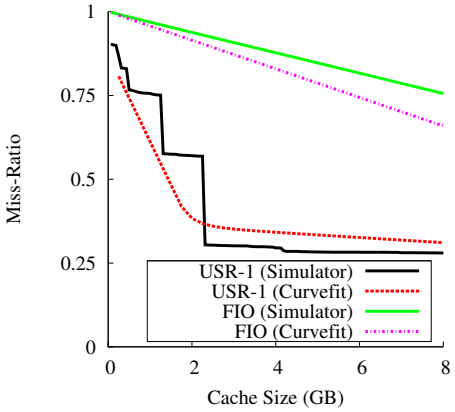


Figure 7: **Miss-Ratio Curves:** *We show the miss-ratio curves for each workload calculated up to when the peak of* USR-1 *is reached (at the 48 minute mark; see Figure 5). It shows that* USR-1 *is more cacheable (hence appropriate for a VCA) than* FIO.

of USR-1 (up to the 48-minute mark). Dynamite determines that USR-1 requires less resources for caching and a cache of 3 GB is needed to offload sufficient I/Os.

Dynamite creates a VCA with 4 GB of memory (some allowance for the VCA's memory requirements). We allocate a large amount of SSD space (unrelated to miss ratio curve) due to the limitations of the VCA version used; the extra space does not impact performance since the working set is small. Following location-affinity, Dynamite creates the VCA on the same compute server as the VM running USR-1 at 55-minute mark (after a sustained SLO violation is experienced).

At this point, the scenarios with and without warmup differ. Without warmup (*Reactive*), the cache becomes effective at the 65-minute mark, and absorbs most of the IOs issued by USR-1, allowing the storage system to satisfy the 5000 IOPS SLO of FIO. The I/O offload results in USR-1 achieving roughly 3000 IOPS between 65 and 75 minutes of the experiment. USR-1 performs better than expected because it is not limited by network bandwidth (due to the collocation of application and VCA).

Cache warmup for the reactive scenario impacts performance negatively (*Reactive-W*). In this particular trace segment, the working set of the prior period of accesses is different from that of the peak workload, thereby causing warmup to simply be an additional burden on the storage system; despite the burden, it completes the peak workload earlier than without a VCA. The negative impact of warmup in this case shows that warmup of the peaking workload may not be useful since the working set of the workload may change significantly at such times. This result may be used to build heuristics that can influence workload selection or warmup decisions.

Proactive Deployment: Proactive deployment is depicted by lines *Proactive* and *Proactive-W* (proactive with warmup) on the graph. For proactive deployment, Dynamite uses predictions of workload peaks to create VCAs *before* SLO violations occur, potentially avoiding violations entirely. As described earlier, we simulate availability of high-quality historical data by using working-set size estimations and zones for warmup for the peak-workload section.

We find that the WSS data for the proactive scenario yields similar miss-ratio curves to the ones for the reactive case. Dynamite creates a VCA with 4 GB of memory for the proactive scenario at the 20-minute mark. The cache becomes effective earlier for the proactive scenario (*Proactive* line) than for the reactive scenario. Further, warmup with reliable data on heavily-accessed zones is extremely useful (*Proactive-W* line). With proactive creation and warmup, Dynamite handles the USR-1 peak with *no* violations of the SLOs of USR-1 and FIO.

7.3 Discussion

While our case study is limited to two main workloads and instantiation of a single VCA, it shows that Dynamite can be very useful for handling SLO violations. Dynamite, especially with proactive instantiation, may even allow the storage stack to avoid SLO violations using compute resources. Further, the ability of Dynamite to handle SLO violations allows the storage system to support two workloads instead of one, thereby providing efficient utilization of resources.

8. CONCLUSION

Storage functionality is no longer confined to dedicated storage servers. As more diverse virtual storage appliances become available, using them similarly to their physical counterparts is an opportunity lost. We can greatly enhance the agility of the data center by using them in dynamic ways.

We use dynamic instantiation of virtual appliances for handling SLO violations. This technique handles workload peaks that occur every few hours or days, a phenomenon that has so far not been addressed adequately. We conduct a caching case study to illustrate dynamic instantiation. To do so, we build Dynamite, a framework that automates cache instantiation. Some of the techniques we have developed for caches may be applicable for dynamic instantiation of other storage appliances as well.

Acknowledgments

We would also like to thank the following people for their guidance and help on dynamically instantiating FlexCache: Joe Caradonna, Philip Clay, Scott Dawkins, Jason Goldschmidt, Jeff Heller, Daniel Holmes, Steve Kleiman, Siddhartha Nandi, Andrew Narver, Shankar Pasupathy, Naresh Patel, Brian Pawlowski, Eric Sirianni, Darrell Suggs, and Narayan Venkat. Finally, we would like to thank Minglong Shao for her feedback on earlier versions of this paper.

9. REFERENCES

[1] Alvarez et al. Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems. *ACM TOCS*, 19(4), 2001.

[2] Amazon Web Services. Amazon ElastiCache, 2011. `http://aws.amazon.com/elasticache/`.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *FAST'02*.

[4] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang. Quickly finding near-optimal storage designs. In *ACM Trans Comput Systems, 23(4):337-374, 2005.*

[5] J. Axboe. fio. `http://freshmeat.net/projects/fio/`.

[6] J. Barr. Animoto - Scaling Through Viral Growth, 2008. http://aws.typepad.com/aws/2008/04/animoto---scali.html.

[7] J. Barr. Scientific Computing with EC2 Spot Instances, 2011. http://aws.typepad.com/aws/2011/09/scientific-computing-with-ec2-spot-instances.html.

[8] L. A. Barroso and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[9] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification, 1995. RFC 1813, http://www.ietf.org/rfc/rfc1813.txt.

[10] D. Chambliss, G. Alvarez, P. Pandey, D. Jadav, J. Xu, and T. Lee. Performance Virtualization for Large-Scale Storage Systems. In *SRDS'03*.

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *SOSP'07*.

[12] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami. Cost Effective Storage using Extent Based Dynamic Tiering. In *FAST'11*.

[13] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional Allocation of Resources for Storage Access. In *FAST'09*.

[14] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar. BASIL: Automated I/O Load Balancing Across Storage Devices. In *FAST'10*.

[15] J. W. Haskins, Jr. and K. Skadron. Minimal Subset Evaluation: Rapid Warm-up for Simulated Hardware State. In *ICCD'01*.

[16] Y. H. Kim, M. D. Hill, and D. A. Wood. Implementing Stack Simulation for Highly-Associative Memories. In *SIGMETRICS'91*.

[17] H. C. Lim, S. Babu, and J. S. Chase. Automated Control for Elastic Storage. In *ICAC'10*, Washington, DC, Jun 2010.

[18] C. Lumb, A. Merchant, and G. Alvarez. Facade: Virtual Storage Devices with Performance Guarantees. In *FAST'03*.

[19] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *VLDB'02*.

[20] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation Techniques for Storage Hierarchies. *IBM Systems Journal*, 9(2), 1970.

[21] M. Mesnier, M. Wachs, R. R. Sambasivan, A. Zheng, and G. R. Ganger. Modeling the Relative Fitness of Storage. In *SIGMETRICS'07*.

[22] D. Narayanan, A. Donnelly, and A. Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. In *FAST'08*.

[23] NetApp, Inc. *NetApp FlexCache*. http://media.netapp.com/documents/tr-3669.pdf.

[24] NetApp, Inc. Data ONTAP Edge, 2012. http://www.netapp.com/us/products/platform-os/data-ontap-edge/.

[25] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems. In *SOCC'11*.

[26] S. Shepler, M. Eisler, and D. Noveck. Network File System (NFS) Version 4 Minor Version 1 Protocol, 2010. RFC 5661, http://tools.ietf.org/html/rfc5661.

[27] T10. SCSI Primary Commands - 4 (SPC-4), Aug. 2011. Working Draft, Project T10/1731-D.

[28] D. K. Tam, R. Azimi, L. Soares, and M. Stumm. RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations. In *ASPLOS'09*.

[29] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger. Stardust: Tracking Activity in a Distributed Storage System. In *SIGMETRICS'06*.

[30] D. Thiébaut, J. L. Wolf, and H. S. Stone. Synthetic Traces for Trace-Driven Simulation of Cache Memories. *IEEE Trans. Computers*, 41(4), 1992.

[31] VMware, Inc. VMware Storage VMotion, 2011. http://www.vmware.com/products/storage-vmotion/overview.html.

[32] VMware, Inc. VMware vSphere Storage Appliance, 2012. http://www.vmware.com/products/datacenter-virtualization/vsphere/vsphere-storage-appliance/overview.html.

[33] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: Performance Insulation for Shared Storage Servers. In *FAST'07*.

[34] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *OSDI'02*.

[35] Y. Wang and A. Merchant. Proportional share scheduling for distributed storage systems. In *FAST'07*.

[36] J. Wilkes. Traveling to Rome: A Retrospective on the Journey. In *R2D2'08*.

[37] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar. Dynamic Tracking of Page Miss Ratio Curve for Memory Management. In *ASPLOS'04*.