Tombolo: Performance Enhancements for Cloud Storage Gateways

Suli Yang, Kiran Srinivasan*, Kishore Udayashankar*, Swetha Krishnan*,
Jingxin Feng*, Yupu Zhang†, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

University of Wisconsin-Madison *NetApp Inc.** *Hewlett Packard Labs†*

Abstract

Object-based cloud storage has been widely adopted for their agility in deploying storage with a very low up-front cost. However, enterprises currently use them to store secondary data and not for expensive primary data. The driving reason is performance; most enterprises conclude that storing primary data in the cloud will not deliver the performance needed to serve typical workloads.

Our analysis of real-world traces shows that certain primary data sets can reside in the cloud with its working set cached locally, using a *cloud gateway* that acts as a caching bridge between local data centers and the cloud. We use a realistic cloud gateway simulator to study the performance and cost of moving such workloads to different cloud backends (like Amazon S3). We find that when equipped with the right techniques, cloud gateways can provide competitive performance and price compared to on-premise storage. We also provide insights on how to build such cloud gateways, especially with respect to caching and prefetching techniques.

1. Introduction

In recent years, the usage of cloud storage has increased tremendously [23]. Cloud storage offers the ability to grow storage with no upfront cost, ease of management, automatic backup and disaster recovery, and data-analytics infrastructure, among other benefits. Recently, IT professionals have identified managing data growth and disaster recovery solutions as the major pain points for enterprise storage [20]. Lack of skilled storage professionals is another commonly identified problem. Moving workloads to the cloud and offloading their management naturally solves the above problems, making it highly attractive to enterprises.

Even though some companies have moved their whole IT infrastructure to the public cloud, some companies would like to keep their computation in-house due to privacy or security concerns, and move only (encrypted) data to the cloud. A *cloud storage gateway* (hereafter cloud gateway) is usually deployed in this case. A cloud gateway exposes standard NAS/SAN protocols (CIFS/NFS/iSCSI) to local clients but is backed by cloud-based object storage. These gateways are typically equipped with caching/prefetching techniques to hide cloud latencies and improve performance.

Currently, enterprises mainly use cloud storage for secondary data such as backup and archival data [18, 38, 48]. This usage is common because secondary data can tolerate WAN latencies as they are rarely accessed. On the other hand, primary data, which are more performance sensitive, continue to occupy the most expensive storage in enterprise data centers, forming the bulk of the cost. However, with the rapid improvement of cloud performance and reduction of cloud cost, we feel it is necessary to revisit such decisions.

In this work, we examine if some of the workloads that require good storage performance but are not highly latency sensitive (termed *tier-2 primary*), can be moved to the cloud. These workloads include document/image repositories, home directories, email files, etc. We look at current cloud performance and prices, and examine how they affect tier-2 workloads. We analyze two real-world traces obtained from production tier-2 primary storage systems containing corporate (MS Office/Access, VM Images, etc.) and engineering (home directories, source code, etc.) datasets. We also build a realistic cloud gateway simulator that has an inline-deduped, log-structured file system with a cloud backend. We replay the workloads in our simulator, with different caching/prefetching algorithms and different cloud backends to evaluate the performance and cost of cloud gateways.

Our observations can be summarized as follows:

1. Typical tier-2 workloads like document/image repositories, home directories, email files, etc., have a relatively small working set and a significant amount of cold data, thus can be cached effectively.
2. A cloud gateway equipped with good caching/prefetching policies can mask cloud latency *even for cache misses*, and reduce tail latency by as much as 70%. For cloud gateways, one should optimize for *partial cache misses* as well as for cache hits.
3. To maximize performance for tier-2 workloads, we need to prefetch based on random access patterns in addition

†This work was done while the author was an intern at NetApp.

to sequential ones. To this end, we also devise a new history-based prefetch algorithm called GRAPH that is suitable for cloud gateway settings and outperforms traditional sequential prefetching schemes.

4. Most importantly, we find that when combined with the right techniques, cloud gateways can deliver performance that is comparable to shared storage performance in local data centers at a lower price. *Cloud gateways are indeed a feasible solution for tier-2 workloads that have reasonable, but not stringent performance requirements*.

The rest of the paper is organized as follows: Section 2 and Section 3 examine the current trends of cloud storage and primary workloads, respectively, and explain why now is a good time to revisit the issue of moving primary workloads to the cloud; Section 4 discusses caching and prefetching techniques suitable for cloud gateways; Section 5 describes the simulator we use to study cloud gateway performance; Section 6 reports our evaluation results and observations; Section 7 describes related work, and Section 8 concludes.

2. Cloud Trends

This section examines current trends in cloud storage, and why now is the time to revisit the issue of moving primary workloads to the cloud.

2.1 Cloud Gateway

In this work, we define a *cloud storage gateway* (or *cloud gateway*) as a hardware or software appliance that enables enterprise applications to access cloud storage over a WAN. In this architecture, applications continue to use traditional NAS or SAN protocols like CIFS or iSCSI, while gateways usually use object-based protocols over HTTP(S) to access cloud storage. Cloud gateways are attractive because they enable high-value data to be kept on-premises for security or performance reasons, while lower-value data can be offloaded to the cloud for cost reduction or ease of management. Commercial gateways include Nasuni [38], Panzura [41], and StorSimple [48]. To act as an effective bridge, a gateway needs additional features: caching and prefetching for performance; deduplication and compression to lower cloud costs; encryption for security; etc.

2.2 Cloud Latency

Cloud latency used to be the limiting factor of the workloads current gateways can support directly. However, this situation is changing.

The awareness of the immense value of reducing latency is increasing, especially from industry giants like Amazon and Google [16, 35, 45]. The networking community is working toward reducing Internet latency, even to speed-of-light delays [46]. Efforts like the Bufferbloat [1] and RITE [4] projects have been actively working on reducing latency over the Internet. ISPs are optimizing for shorter network latency [21].

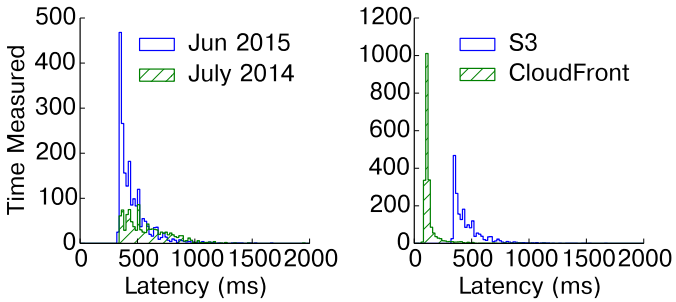

Figure 1: **Cloud Latency Distribution (Amazon S3)**. Measured using 64KB objects to the nearest AWS (Amazon East) site for GET operations. S3 objects of different sizes were evaluated. The nature of the latency distribution remains the same, while the average latency increases roughly linearly with the object size. The experiments were repeated at different times of the day and on different days in a week, with similar results.

In the meanwhile, cloud storage providers are also working to reduce latency and to cater to more workloads. For example, Amazon CloudFront, a content delivery network service tailored to S3 objects, is an effort in this direction [7, 8]. CloudFront is steadily increasing the number of edge locations, starting from six in 2007 to around sixty now. In the coming years, the number is expected to grow to several hundred. This trend implies that the likelihood of finding an edge location nearby for most enterprise data centers is very high. Therefore, we can expect latencies to become lower and more stable.

WAN acceleration techniques can be used to further reduce latency too, and are popular in academic and commercial contexts (e.g. LBFS [37] and Riverbed [43]). They typically involve optimizations to make an inefficient data transfer protocol function better with fewer WAN round trips. Given the dependence on protocols, current techniques have a footprint on both sides of the network connection.

In Figure 1 (left) we compare the cloud latency measured at different times (2014 and 2015) from the same server within the enterprise network. We can see that over time, the latency variability decreases, giving us lower and more stable cloud access time. In Figure 1 (right) we compare the latency of accessing Amazon S3 objects directly versus accessing the same objects via Amazon CloudFront. We can see that CloudFront reduces latency significantly, making it more attractive for moving data to the cloud.

Given the above trends, we believe that the feasibility of moving tier-2 primary workloads to the cloud (via gateways) is becoming clear.

2.3 Cloud Storage Costs

The cost of cloud storage is declining rapidly. Figure 2 shows the storage price change from three major cloud storage providers over the past 10 years. We can see that the

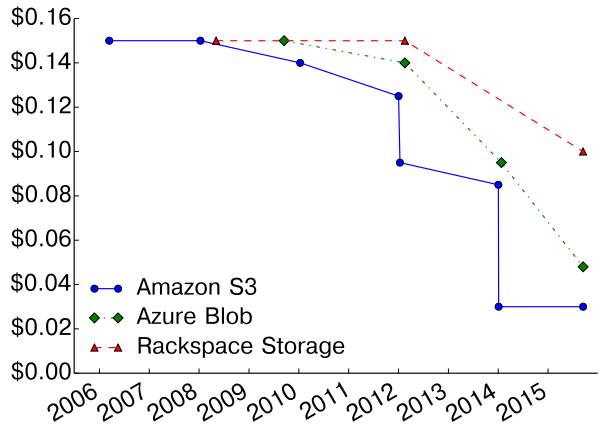

Figure 2: **Cloud Price Declines**. Monthly cost per GB of cloud storage. The prices shown here are for geo-replicated data and for the first TB of storage. More detailed pricing information can be found in the providers' websites.

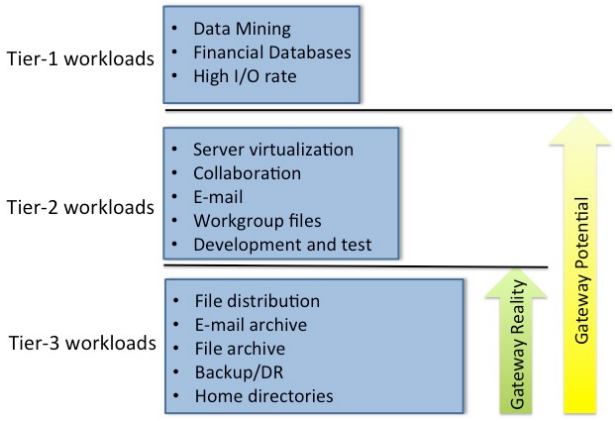

Figure 3: **Workload Categorization**. Source: Gartner

monthly cost of storing data in public clouds dropped as much as 75% from 2012, and the trend is likely to continue. The price of cloud storage requests is also droping by large margins. For example, Amazon dropped the S3 storage `GET` request prices by 60% and `PUT` request prices by 50% in 2013. Because of the large price drop, the economics of using cloud storage needs to be reexamined. Later (§ 6.3) we will show that when combined with a gateway with effective caching, the cost of storing data in the cloud is indeed comparable to or lower than on-premise storage.

3. Workload Trends

From a workload standpoint, Gartner classifies workloads that can leverage the gateway into two categories: *gateway reality* and *gateway potential* [22]. As shown in Figure 3, tier-3 workloads have already moved to the cloud via gateways, whereas tier-1 workloads are so highly latency sensitive that they are not expected to move to the cloud. However, tier-2 primary workloads could be moved to the cloud via gateways in the near future. Because primary workloads

	Corporate	Engineering
Clients	5261	2654
Duration (Day)	42	38
Read Requests	19,876,155	23,818,465
Write Requests	3,968,452	4,416,026
Data Read (GB)	203.8	192.1
Data Written (GB)	119.9	87.2
Data Growth (GB)	82.3	63.7

Table 1: **Summary of Workload Statistics**.

are hosted on the expensive enterprise storage systems, the savings in maintaining only a smaller cache of the entire data would be enormous. Moreover, the costs of maintaining the primary data (and its secondary copies) would be delegated completely to the cloud. We thus investigate the feasibility of moving tier-2 workloads to the cloud.

3.1 Tier-2 Workloads

To evaluate the potential of moving tier-2 workloads to the cloud, we use two publicly available real-world CIFS traces obtained from an enterprise primary storage system [32].

One trace was collected from a mid-range file server in the corporate data center that was used by roughly 1,000 employees in marketing, sales and finace departments. It mainly contains Microsoft Office, Microsoft Access data, and virtual machine images, and represents about 3 TB of total storage. We call this trace the Corporate dataset (Corp).

The other trace was collected from a high-end file server deployed in the engineering data center used by roughly 500 engineers. It mainly contains home directories, source code, and build data, and represents about 19 TB of storage. We call this trace the Engineering dataset (Eng).

Table 1 shows some high-level statistics of the two workloads. Note that these statistics are collected at the block level, underneath the file system and deduplication engine (§ 5.2). File-level statistics are available in the original publication [32]. One key observation from Table 1 is that there is significant data growth on both datasets: on average about 70% of the data written are new data transferred into the storage system (even after deduplication). This fast growth rate indicates that the "pay-as-you-go" model offered by cloud storage may be more suitable than having to provide excess capacity upfront. We discuss the detailed cloud cost incurred by a cloud gateway later (§ 6.3).

Figure 4 shows the working set sizes of both datasets on a daily basis. We can see that even though the total storage size is large (Terabyte scale), the working set of the workload is quite small and can easily fit into a local cache, making a cloud gateway feasible.

We now investigate the sequential access patterns of the workloads. Figure 5 shows the distribution of sequential run sizes, both in terms of number of runs and number of 4K blocks (a sequential run is defined as a series of sequential block accesses). On one hand, a significant portion of the accesses are sequential, with run sizes as large as 10,000

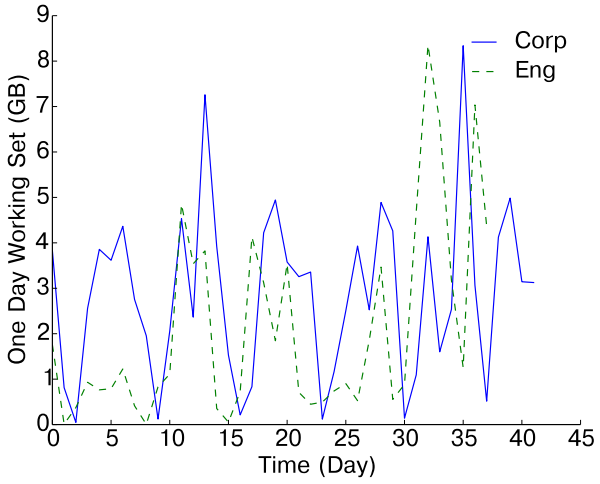

Figure 4: **Working Set Size**.

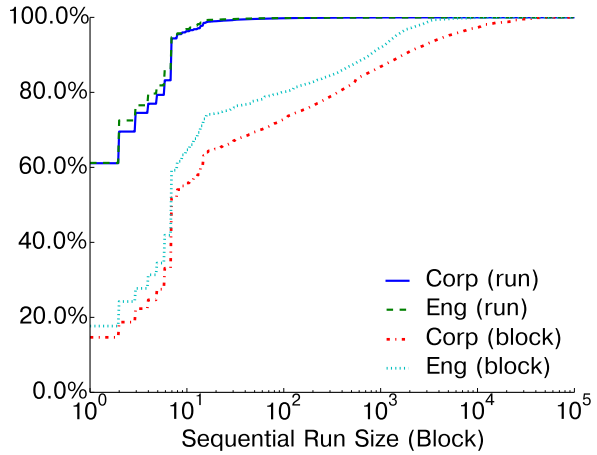

Figure 5: **Sequential Run Size CDF**.

blocks, indicating the potential of using sequentiality-based prefetching techniques (§ 4.2.1) to reduce access latency; on the other hand, more than 60% of the blocks are still accessed in runs of less than 10 blocks. Given the long latency of fetching data from the cloud, sequential prefetching is unlikely to help in these cases, so other prefetching techniques, e.g., history-based prefetching (§ 4.2.2) may be necessary to further reduce latency, especially the tail latency. Our later experiments confirm this observation (§ 6.2).

3.2 SLO Requirements

In order to answer the question "*what kind of storage performance is required to support tier-2 workloads?*", we look at some storage systems deployed in *local* data centers and the performance guarantees they offer. We will concentrate on latency as this is typically the bottleneck of cloud gateways.

PriorityMeister [53] claims to be a state-of-the-art approach to provide tail latency guarantees in a shared data center as of 2014. Table 2 reports the tail latency it offers to several primary workloads when running in conjunction with a non-latency-sensitive file-copy workload. The 90th percentile tail latency could be as long as 200 ms. When

	90%	95%	99%
Display Ads Production	91 ms	129 ms	218 ms
MSN Storage Production	126 ms	170 ms	258 ms
LiveMaps Production	200 ms	300 ms	509 ms

Table 2: **PriorityMeister Tail Latency**. Workloads share disks with a throughput-oriented, non-latency-sensitive workload that represents a file copy.

competing for resources with other bursty, latency-sensitive workloads, the tail latency guarantees PriorityMeister provides are even weaker. For example, the 90th tail latency of an Exchange server workload is around 700 ms when running concurrently with a LiveMaps production workload.

Cloud providers, such as Google, Amazon, or Azure, are another comparison point. When accessing data from a virtual machine hosted by the same vendor in the *same region*, the 99th tail latency of the time to first byte (TTFB) was reported to be more than 100 ms in Dec. 2015 [15].

Since most primary workloads are served in these (shared) local data centers, they must be able to tolerate the above latencies of a few hundred milliseconds. This also agrees with our observation that even though tier-2 workloads require good performance, occational long latency can be tolerated by applications. For example, the CIFS protocol, meant for tier-2 workloads, can tolerate a latency of up to 15 seconds in the path of retreval [3]. We show later (§ 6.2) that our cloud gateway is able to match such SLOs, where we offer overall good latencies, and the tail latencies are kept in the sub-second level.

4. Caching/Prefetching Techniques

Mitigating WAN latency is a key concern for gateways. In this section we survey the latency-mitigation techniques that are most relevant to cloud storage gateways: caching and prefetching. These techniques are commonly used in storage systems, but cloud gateways have some unique characteristics compared to traditional caching/prefetching at the page-cache level. First, within cloud gateways, more complex algorithms that come with overheads in the form of extra computation or extra metadata are acceptable, as long as it is possible to avoid even a few cloud I/Os. Second, besides performance, we should also minimize monetary cost. Based on these observation, we adapt existing algorithms, as well as devise new ones. Table 3 gives a summary of the techniques we investigate. We discuss them in detail below.

4.1 Caching Techniques

Caching is one of the most prevalent techniques to improve performance and reduce latency, and is widely used in cloud gateways. Looking beyond traditional cache replacement policies like *LRU* [11], we investigate newer adaptive algorithms like *SARC* [25] in the gateway context.

SARC (Sequential prefetching in Adaptive Replacement Cache [25]) partitions the cache between random and sequential access streams in two separate LRU lists, SEQ and

	Cache	Prefetch	Origin
LRU_SEQ	LRU	Fixed-Size	[25]
LRU_AMP	LRU	AMP	[24]
LRU_GRAPH	LRU	History-Based + AMP	*new*
SARC_SEQ	SARC	Fixed-Size	[25]
SARC_AMP	SARC	AMP	*new*
SARC_GRAPH	SARC	History-Based + AMP	*new*

Table 3: **Summary of Techniques**. Naming convention is *<caching algorithm>_<prefetching algorithm>*.

RANDOM. The algorithm trades cache space between SEQ and RANDOM dynamically by maintaining the marginal utility of allocating additional cache space to each list, thus minimizing the overall cache-miss rate. Because prefetching is performed only for accesses in the SEQ list, such separation and adaptation avoids thrashing where more valuable demand-paged random blocks are replaced with less precious prefetched blocks, or prefetched blocks are replaced too early before they are used.

We implement both LRU and SARC in our simulator. Prior evaluations [25] have coupled SARC with simple, fixed-degree sequential prefetching; we combine SARC with different prefetch techniques, as described below.

4.2 Prefetching

High latency is the main obstacle of moving data to the cloud. Prefetching, as an effective technique in masking latency, is thus important. Prefetching in cloud gateways is different from traditional prefetching because aside from performance, wasteful prefetches are a bigger concern. Most cloud providers charge only for data transfer out of the cloud, so reducing unnecessary fetches is key to reduce costs. We classify prefetching techniques into two categories: sequentiality-based and history-based.

4.2.1 Sequential Prefetch

Sequential prefetch exploits the spatial locality of access patterns and is widely used in storage systems [29, 49]. With a sequential prefetch algorithm, the system tries to identify sequentially-accessed streams and then performs readahead on those streams. We find sequential prefetching attractive for cloud gateways because it has a high predictive accuracy (avoiding wasteful prefetch) and is easy to implement.

The two most important aspects of a sequential prefetch algorithm are the prefetch time (*when* to prefetch) and prefetch degree (*how much* to prefetch). Because of the long latency of accessing cloud storage, simple synchronous prefetch, where prefetching only happens upon a cache miss, is unlikely to suffice. Instead, the likelihood of accessing certain blocks has to be identified early enough to allow sufficient time to fetch data from the cloud. Also, the degree of prefetching needs to be continuously adapted to minimize wasteful prefetch.

Based on the above two considerations, we find *AMP* (Adaptive Multi-stream Prefetching [24]) attractive. AMP keeps track of each sequential stream it identifies, and determines the prefetch time and prefetch degree based on the rate at which clients are accessing each stream. If on the completion of a prefetch a read is already waiting for the first block being prefetched, AMP increases the trigger distance (the distance between the block that triggers a prefetch and the last block in the stream) for that stream; if the last block of a prefetch set is evicted before it is accessed, AMP reduces the prefetch degree for that stream.

Tap [33] is another sequential prefetch algorithm. It uses a table to detect sequential access patterns in the I/O workload and to dynamically determine the optimal prefetch cache size. However, it assumes a small cache size and underperforms other algorithms with larger cache capacities, so we find it unsuitable in a cloud-gateway context.

We implement AMP as well as synchronous, fixed-degree sequential prefetch in our simulator. The original AMP does not distinguish between sequential and random accesses. To further reduce wasted prefetch, we combine AMP with SARC, and perform prefetch only for sequential accesses.

4.2.2 History-based Prefetch

Sequential prefetching is beneficial only when successive I/O accesses are contiguous; it does not improve performance for random access patterns. Even for sequential streams, when there are jumps or discontinuities within the stream, such techniques are inadequate. There are techniques that use past I/O access history for prefetching [10, 26, 30, 31, 47], but they have not been used in enterprise storage systems due to their complexity and metadata overheads [24]. However, in our context, any extra complexity or overhead is acceptable provided that it results in appropriate benefits.

Griffioen et. al. [26] builds a *probability graph* to represent file access patterns and facilitate whole-file level prefetch. In their probability graph, a node represents a file and an arc from node `A` to node `B` represents the probability of file `B` being opened soon after file `A`. Amer et al. refines this idea by using recency information in addition to probability to build the access graph [9]. Nexus [27] constructs a similar relationship graph using a large look-ahead history window size and captures access patterns with larger gaps in between when doing mobile prefetching. These techniques all work on a per-file basis.

At the block level, DiskSeen uses a block table to track access trails and tries to match the current trail to historical trails in order to identify prefetchable blocks [19]. However, it exploits the underlying disk layout information for prefetching, rendering it unsuitable for cloud gateways. C-Miner [34] uses frequent sequence mining to discover block correlation rules and prefetches based on the discovered rules, but it is mostly an offline approach.

While all these techniques are relevant, none is a good fit for the specific needs of a cloud gateway (most of them are built for buffer-cache prefetching originally). Inspired by the above ideas, we devise our own access-graph-based prefetch

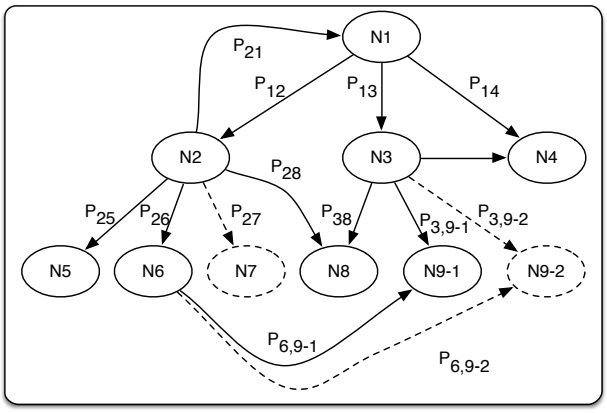

Figure 6: **History Access Pattern Graph**.

algorithm *GRAPH* for cloud gateways. Our method of building the access graph is largely borrowed from [26], except that GRAPH works at the block layer. We use spatial information and recency to enhance our access graph, and apply balanced expansion [10] for traversing I/O access history. Our graph prefetch algorithm is designed to capture random access patterns, and works in conjunction with sequential prefetching techniques to maximize prefetch opportunities. The details of GRAPH are described below.

4.2.3 The GRAPH Prefetch Algorithm

Graph Construction: Our graph maintains the client I/O history in terms of *block ranges* (BRs), as shown in Figure 6. Each node in the graph is a contiguous range of blocks. We choose to maintain the graph on the block level (instead of the file level) to capture correlations between blocks across different files, and metadata accesses. Using contiguous block ranges instead of single blocks helps keep the graph smaller without losing important information. Each outgoing edge E from node N_1 in Figure 6 represents an access pattern of N_1 followed immediately by N_x (x = 2, 3, or 4). Each such access pattern is associated with a likelihood, which is our expectation of how likely N_x is to be accessed in the near future given that N_1 is accessed.

Upon each I/O request, we find the block range node N_1 that this request accesses, and also the block range node N_0 which its previous request accessed. If N_1 is a new block range, then a new node is added to the graph. The block ranges do not overlap, and any possible overlap is removed by splitting the nodes as appropriate. If there is already an edge to represent the access pattern of N_1 after N_0, the likelihood of this access pattern is updated. If there has not been such an access pattern before, an edge from N_0 to N_1 is added to the graph, and its likelihood is assigned.

We look only at the previous I/O request and not any further in the access history. This loss of information could cost us some prefetch opportunities. For example, a long chain of repeated access, say $BR_1 \rightarrow BR_2 \rightarrow BR_3 \rightarrow BR_4$, would appear no different if $BR_1 \rightarrow BR_2$, $BR_2 \rightarrow BR_3$, and $BR_3 \rightarrow BR_4$ happened separately. Any repeated access pattern with a semantic distance larger than 1, e.g., $A \rightarrow X \rightarrow B$, where A is always followed by a random block X then by B, will not be discovered. However, this keeps our graph size manageable.

The formula we use to calculate the likelihood of accessing BR N_Y after BR N_X, L_{X-Y}, is:

If N_Y is sequentially adjacent to N_X:

$$L_{X-Y} = \begin{cases} 1 & \text{No accesses of } N_X \text{ followed by } N_Y \text{ yet} \\ \frac{1+C_{X-Y}}{1+C_{X-all}} & \text{Otherwise} \end{cases} \quad (1)$$

If N_Y is not sequentially adjacent to N_X:

$$L_{X-Y} = \begin{cases} 0 & \text{No accesses of } N_X \text{ followed by } N_Y \text{ yet} \\ \frac{C_{X-Y}}{C_{X-all}} & \text{Otherwise} \end{cases} \quad (2)$$

where C_{X-Y} is the recent access count from N_X to N_Y, and C_{X-all} is the total recent access count from N_X to all neighbors (recorded accesses are expired after a configurable threshold). In the first case of (2), there would not be an edge.

The intuiton behind the formula is to use the probability of N_Y following N_X (instead of any other immediate child node of N_X) as the likelihood. Sequential access patterns are given an initial likelihood of 1, but their weight fades away when random access patterns are observed.

Our experiments show that it is important to incorporate sequentiality information into likelihood. Likelihood based on probability alone is not a good indicator when there is insufficient access history for a particular block range. In our graph, only the likelihood of an immediate adjacent access pattern is explicitly given. We calculate the likelihood between non-directly-connected nodes by multiplying the likelihood along all the edges of the shortest path between the two nodes to help us prefetch deeper.

Graph Traversal: Upon a hit to a block range, i.e., a node N, the graph is traversed from N to generate a set of prefetch candidate nodes. Standard graph traversals such as depth-first-search (DFS) and breadth-first-search (BFS) can be used. We use another technique, balanced expansion [10], to find the best candidates. In balanced expansion, nodes that are more likely to be accessed are always chosen over nodes that are less likely. We start traversal from an accessed node, N_1, and maintain two sets: the traverse set T, which initially contains the immediate successors (those connected by a direct outgoing edge) of node N_1, and the candidate set C, which is initially empty. We then repeat the following process: select node N from T which has the highest likelihood of being accessed from node N_1. The likelihood is calculated by multiplying the probabilities along all the edges of the shortest path from N_1 to N. We then remove N from T and add N to C. Then we add all the immediate successors of N into T, and again select the highest likelihood node in T and move it to C. We repeat this process until the likelihood is below a certain threshold. In the end, C contains all the nodes to be prefetched. A proof that balanced expansion is the best mechanism based on successor prediction is

straightforward [10]. In our experiments, we also found that balanced expansion performs better than BFS or DFS.

Our algorithm also uses a graph-aware variant of AMP to set the size limit of the prefetch candidate set and the trigger distance. When some prefetched blocks are not accessed before being evicted, there are two possibilities: we made wrong prefetches and these blocks would not be accessed (in the near future), or we prefetched too much and these blocks are evicted before they would be accessed. The original AMP only considers the latter possibility because a sequential access stream is highly likely to extend. However, in our case, as there are multiple possible paths from a given BR node in our history graph, we need to differentiate between these two scenarios. We achieve this by keeping track of both the recently accessed blocks and the recently evicted blocks. If a block is being evicted, we decrease the prefetch degree in case it is later accessed as we had predicted. Whereas for a wrong prefetch decision, we just retain the previous prefetch degree.

Optimizations: Maintaining the complete access history necessary for a naive history-based approach is prohibitively expensive, so we optimize the naive scheme based on two observations. First, tier-2 workloads are dominated by sequential streams, which are satisfied by sequential prefetching. Second, a history-based approach is effective only for correlated random accesses and for jumps between related sequential streams. Therefore, in our technique, we use the second observation to populate the I/O history graph *only with random accesses*. The likelihood of sequential access patterns would fade away faster under this optimazation, as C_{X-Y} in formula (1) is always considered zero. This reduces our graph size considerably, making traversals faster. With this change, prefetching is done as follows: we check whether the current I/O is sequential with any known streams in the local cache. If so, we use AMP, otherwise we use the graph to prefetch. In our experiments, we find that this reduces the metadata size by 80% and runtime by 90% compared to the naive approach, with no loss in hit ratio or prefetch efficiency.

As discussed before, the likelihood between non-directly connected nodes is calculated by multiplying the likelihood along all the edges of the shortest path between the two nodes. This is a rather expensive operation since it requires traversing the graph to find the shortest path. To improve performance, we cache the likelihood values between nodes for 60 seconds for reuse in the near future.

Combining SARC with GRAPH: To take advantage of SARC's effective cache replacement policy, we combine GRAPH and SARC, resulting in SARC_GRAPH. We use AMP to prefetch for sequential access streams and GRAPH for random ones. The balance of cache space between random and sequential accesses is dynamically adapted to minimize cache misses.

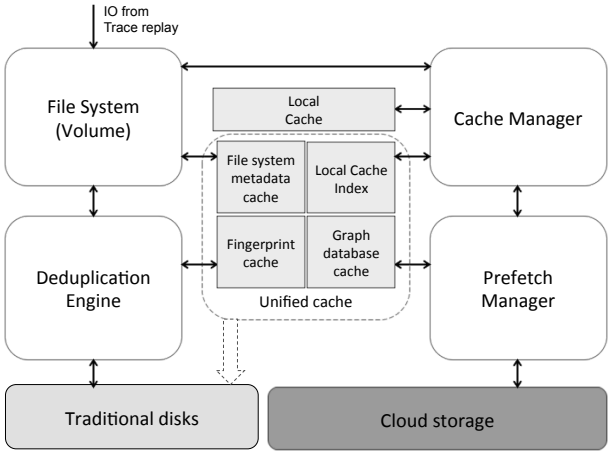

Figure 7: **Cloud Gateway Simulator**.

5. Simulator

As described before, tier-2 primary workloads (Figure 3) have performance requirements. Typically, enterprise storage systems co-resident with clients can satisfy their requirements. To evaluate our techniques on tier-2 workloads with data residing in cloud storage, we built a cloud gateway simulator (Figure 7) with subsystems that closely resemble an enterprise-class storage system. The simulator is designed as a software appliance with a gateway cache on disk and a BerkeleyDB [40] key value store simulating cloud storage. There are two important components in the simulator: the trace replay engine and the simulator file system. The file system in turn has multiple subsystems and each of them maintains its own on-disk persistent metadata and associated in-memory cache. We also collect statistics at each stage of the simulator to evaluate the performance of individual components. We now describe these components.

5.1 Trace Replay Engine

An accurate trace replay is critical to evaluate performance of a file system cache. Our replay engine is designed to maintain strict I/O ordering using a time-based event simulation technique [42]. Specifically, during I/O processing, asynchronous interactions between modules in the file system are represented as different events in the replay engine. Such interactions typically involve queuing delay, which is accurately accounted for in the engine. Apart from I/O processing events, periodic flush of cached (dirty) data to underlying storage media is simulated using synchronous and asynchronous data flush events. Although queuing delay and wait times are captured during event processing, we have ignored processing time, as we found it to be small compared to disk access and WAN latencies. In our simulation, the traces are replayed in succession though the actual traces were captured over a long period.

5.2 Simulator File System

The simulator file system has the following subsystems: Deduplication Engine, Cache Manager, Cloud Driver Mod-

ule, Prefetch Manager, and Unified Cache. The simulator presents its file system as a data container, i.e., a volume, to the trace replay engine. It maintains an allocation bitmap and assigns a new physical volume block number (PVBN) for each incoming write block. To model a log-structured file system with infinite capacity backed by cloud storage, the PVBN is a monotonically increasing value. Once a PVBN is deallocated, it is not reused. In addition, incoming reads (represented as {*filename, offset*}) are mapped by the file system to the corresponding PVBN. This map is maintained in a BerkeleyDB [40] table that associates a PVBN with each {*filename, offset*} tuple. The above mentioned subsystems are described next.

Deduplication Engine: Deduplicating data before sending it to the cloud is important: on the cloud side, storing less data can reduce cloud storage costs and save bandwidth; on the local side, deduplication enables caches in the data path to be more efficient. Thus, any simulation of a cloud gateway should take deduplication into account.

Our simulator uses an inline deduplication engine, which creates a SHA256 digest of every block sent to the cloud. These digests are persisted in an on-disk fingerprint database and mapped to the corresponding PVBN. A small in-memory fingerprint database cache is maintained to improve lookups in the I/O path. For all newly written blocks, we first check against this fingerprint database to eliminate duplicates. Along with the fingerprint database, to ensure consistency after deletes, for each unique block we keep the count of its references in a separate file, called the *refcount-file*.

Cache Manager: The simulator maintains a cache for the workload's working set, which is managed by the cache manager. As cloud storage is elastic we expect the data stored to be potentially on the order of petabytes. However, we expect the working set of these workloads to be a smaller fraction of this size (though still much too large to cache in memory). The gateway thus maintains a disk-based cache, implemented as a file called the *local-cache* whose size is configurable. For simplicity, metadata for blocks stored in the local-cache is maintained in an in-memory data structure. A newly cached block is represented by its data written to the local-cache file at a specific offset, which is used to index this block and is referred to as the LCBN (Local Cache Block Number). In addition, we have designed a pluggable interface within the cache manager to experiment with different cache replacement policies. To support writes, the cache manager is tasked with flushing dirty cache blocks to the cloud. These dirty blocks are flushed either periodically or to free up cache space.

Cloud Driver module: This subsystem is created to flush dirty blocks and to read blocks from the cloud. The latencies of cloud accesses are drawn from the actual latency distribution as measured against a given cloud backend. To provide better throughput, the cloud driver packs a fixed, but configurable, number of data blocks into each cloud object. Each such cloud object is associated with a cloud key that is generated from the SHA256 digests of its constituent blocks. The mapping from PVBN to cloud object key is maintained in a persistent database called the *Cloud-DB*. In our simulator, cloud storage is simulated by a BerkeleyDB database that is accessed via a key-value interface similar to actual cloud interfaces. To implement this, the cloud driver creates a *B-tree hash* instance to store unique cloud objects.

Prefetch Manager: The purpose of this module is to enable prefetching from the cloud. As with the cache manager, the prefetch manager module provides a pluggable interface to experiment with various prefetching techniques. It provides both synchronous and asynchronous modes to prefetch blocks into local-cache from the cloud via the cloud driver. During simulation, due to the large latencies of the cloud, duplicate read or prefetch requests might get queued at the prefetch manager waiting for inflight blocks. Such requests will result in suspending the corresponding I/O until the cloud driver responds.

The prefetch manager also maintains all the metadata necessary to make prefetch decisions. For sequential prefetch algorithms, this metadata is small, e.g., SARC maintains a bit on each cached block to denote sequential or random access. However, for history-based algorithms, the metadata is represented as a graph and is much larger. In those cases, the graph is maintained as a combination of an in-memory cache and an on-disk structure. Moreover, this graph needs to be stored on-disk in a compact form with efficient insertions and traversals. Modern graph databases [5, 36, 39] seem to be appropriate for this task, as they are optimized for traversal and minimize disk accesses. In our implementation, we use DEX [36], but other graph databases could be accommodated as well. We also build additional indices based on interval trees to quickly query BR nodes based on block ranges, or which BR node holds a particular block.

Unified Cache: All persistent metadata used by different subsystems in the simulator is stored on disk, as in an enterprise storage system. We maintain an in-memory cache of these metadata structures for better performance. Care has also been taken to allocate a fixed-size cache for BerkeleyDB and graph database software.

5.3 Summary

The architecture of our simulator closely resembles production storage gateway systems, such as AWS Storage Gateway by Amazon [6], or the cloud storage gateway by Brocade [44]. Compression and encryption would work similarly to the deduplication engine, maintaining any associated metadata locally and utilizing the unified cache.

We made a few design choices, most notably working at the block level instead of the file level. There are several considerations. First, the gateway needs to work with WAFL, a log structure file system that appends new blocks at the end and retains the "deleted" blocks for snapshots [28]. By work-

ing at the block level, the contiguous blocks corresponding to the older snapshots can be packed into an object efficiently and be moved to the cloud irrespective of the file boundaries. This behavior is especially useful when one has many small files. Second, many files for tier-2 workloads are virtual disk images that represent the entire file system for the VMs and are large. Users typically just modify a portion of the file, and transferring the entire image to the cloud is not feasible. At the block level we can keep the cold parts of the file in the cloud. Third, we map only contiguous block ranges to objects. Since WAFL tends to have high sequentiality, the metadata is kept small and efficient to manage. In summary, our design allows maximum flexibility and low overhead.

Other popular commercial cloud storage gateways, such as AWS Storage Gateway [6] or EMC Cloud Array [12], work at the block level too.

6. Experimental Evaluation

In this section, we evaluate the performance and cost of cloud gateways when combined with different caching and prefetching techniques, and using different cloud backends. We try to answer the following questions: *Can tier-2 workloads be moved to the cloud? What kind of techniques are needed to enable moving data to the cloud? At what cost?*

6.1 Experimental Setup

Table 3 gives a summary of the combination of techniques we investigate. In our configuration, each cloud object packs 64 4KB blocks. We use cloud latency distribution measued from two different cloud storage backends: Amazon S3 and Amazon CloudFront+S3, because they offer distinctly different latency characteristics [7, 8]. The numbers we report here were collected in 2014. We do notice that cloud latency has improved over the past year, but while the overall latency is lower, the nature of our results remain the same.

We evaluate three different local cache size settings: 30 GB, 60 GB and 90 GB. For the Corporate dataset, this cache size represents 1% to 3% of the total data size, and 10% to 30% of the total data accessed in the 42-day trace duration. For the Engineering dataset, it represents 0.16% - 0.48% of the total data size, and again 10% to 30% of the total data accessed.

6.2 Cache Miss and I/O Latency

Cache Miss: I/O latency is one of the key metrics for storage systems, and also the main concern of moving data to the cloud. The main goal of different caching/prefetching algorithms is to reduce the number of cache misses, thus masking the long latencies of accessing the cloud. Hence the cache-miss ratio is an important performance indicator for cloud gateways. The cache manager in our simulator collects counts of cache hits and misses to compute this metric.

In Figure 8, we show the cache-miss ratio for different algorithms with varying local cache sizes in a cloud gateway context. Adaptive caching (SARC_SEQ) reduces cache misses for both datasets, though more significantly for Corp. Adaptive prefetching (LRU_AMP) reduces cache misses for Corp, which has a larger data footprint, but not for Eng. In general, we observe that AMP performs better with a small cache compared to the workload's data footprint. However, we always achieve better results when combining adaptive caching and adaptive prefetching together (SARC_AMP). Relative to LRU_SEQ, SARC_AMP reduces the miss ratio from 21% to 13% for the Corp dataset when using a 30 GB local cache. When using a history graph to capture random access patterns and performing prefetch based on these patterns in addition to sequential prefetching, we can reduce the cache-miss ratio even further. For both datasets, the cache misses can be reduced to less than 10% with graph-based prefetching techniques, and as low as 6% with SARC_GRAPH.

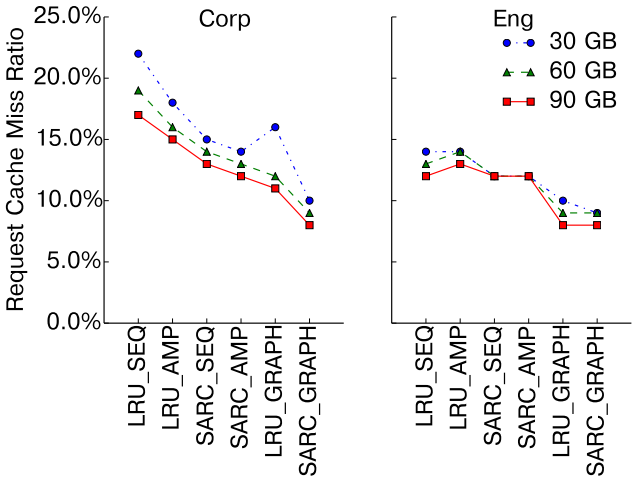

Figure 8: **Cache Miss Ratio**. The numbers shown here were collected using CloudFront+S3 as the cloud storage backend. The cache miss ratios are similar when using S3 directly (not shown here).

Algorithms	90%	95%	99%
LRU_SEQ	935 ms	1585 ms	2196 ms
LRU_AMP	834 ms	1475 ms	2165 ms
SARC_SEQ	745 ms	1335 ms	2115 ms
SARC_AMP	705 ms	1255 ms	2095 ms
LRU_GRAPH	644 ms	1156 ms	2075 ms
SARC_GRAPH	33 ms	885 ms	1976 ms
Cloud	2105 ms	2322 ms	2865 ms

Table 4: **Tail Latency (S3)**, Corp, 90 GB cache.

I/O Latency: Next we study the end-to-end I/O request latency in a cloud-gateway setting, and investigate how cache misses impact the latencies ultimately seen by the client. We will concentrate our discussion on read requests since writes can always be buffered locally and flushed to the cloud in an asynchronous fashion. There are three cases when a client issues a read request to a cloud gateway:

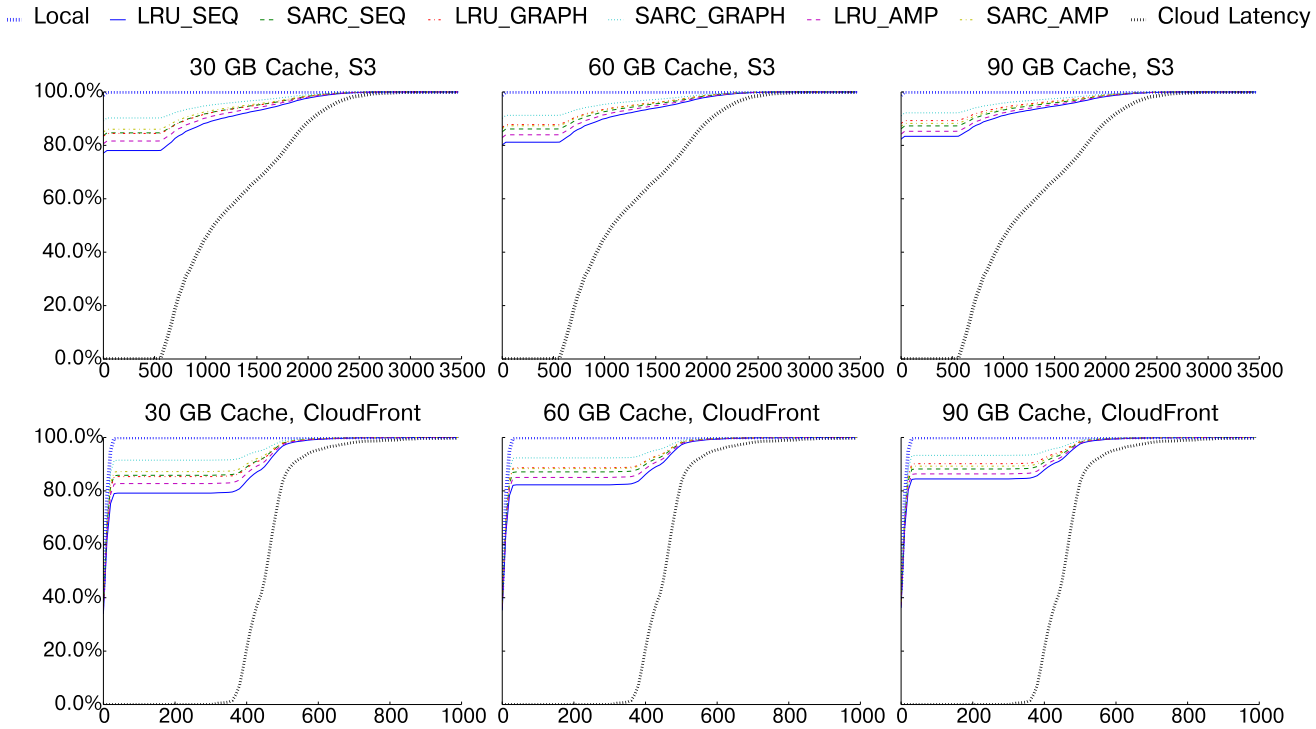

Figure 9: **Latency CDF, Corp Dataset**. X-axis denotes request latency in milliseconds. *Cloud Latency* shows the latency distribution of accessing cloud storage directly. *Local* shows the latency distribution if all accesses are local. For all but one case (LRU_SEQ in 30 GB cache), 80% tail latency is less than 40 ms.

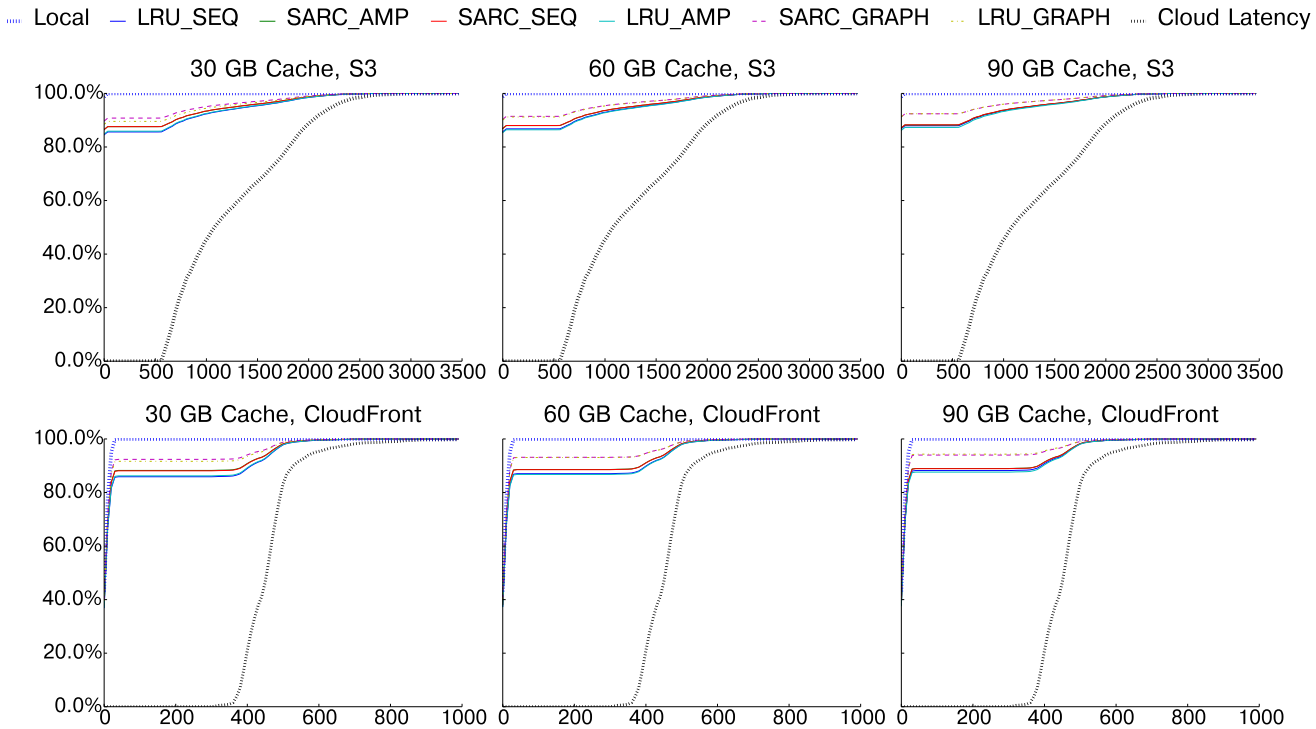

Figure 10: **Latency CDF, Eng Dataset**. Same as Figure 9

Algorithms	90%	95%	99%
LRU_SEQ	435 ms	485 ms	564 ms
LRU_AMP	414 ms	475 ms	556 ms
SARC_SEQ	405 ms	475 ms	545 ms
SARC_AMP	386 ms	465 ms	534 ms
LRU_GRAPH	40 ms	465 ms	534 ms
SARC_GRAPH	33 ms	414 ms	515 ms
Cloud	539 ms	593 ms	805 ms

Table 5: **Tail latency (CloudFront)**, Corp, 90 GB cache.

1. *Cache hit*: The requested block is already in the local cache; only the local disk latency is incurred.
2. *Complete cache miss*: The requested block is not present in the local cache, nor is it being currently fetched. A fetch to the cloud must be initiated, and full cloud latency is observed by the client.
3. *Partial cache miss:* The requested block is not present in cache, but is currently being fetched. The fetching of the data in question could either be triggered by the prefetch algorithm, or by the cache misses of other requests. In this case, the cloud latency is partially masked from the client. The exact latency the client experiences depends on the timing of the fetch.

Our simulator accounts for all three cases and simulates latency faithfully in each case. We simulate cloud latencies by following the latency distribution we measured against the cloud storage backend (in our case, S3 and CloudFront). The local-cache latency is derived from the latency distribution of randomly accessing a 500 GB hard disk drive. The I/O request may also incur metadata accesses, the latency of which is not taken into account. We believe that metadata should be kept in low-latency media (e.g, high-end SSDs) and well cached; the latency introduced should thus be small.

The CDF of the simulated per-request latency under different caching and prefetching settings are shown in Figures 9 and 10. We also list the 90th, 95th and 99th percentile tail latencies of the Corporate dataset for the 90 GB cache setting in Tables 4 - 5. The Engineering dataset and other cache settings show similar characteristics for tail latencies; in particular, the 90th percentile tail latency when using SARC_GRAPH is always under 40 ms with different datasets and under different cache settings.

First, we notice that history-based prefetch (LRU_GRAPH and SARC_GRAPH) consistently outperforms sequential prefetch for both datasets and in different settings. For the Engineering dataset (Figure 10), the four sequential prefetch algorithms show similar performance, indicating a limitation of sequential prefetch, but the GRAPH algorithms are able to deliver better performance by capturing random access patterns. This result agrees with our earlier observation of primary workloads (Figure 5), which suggests significant random access. We conclude that *history-based prefetch is necessary to improve the performance of cloud gateways for tier-2 primary workloads*.

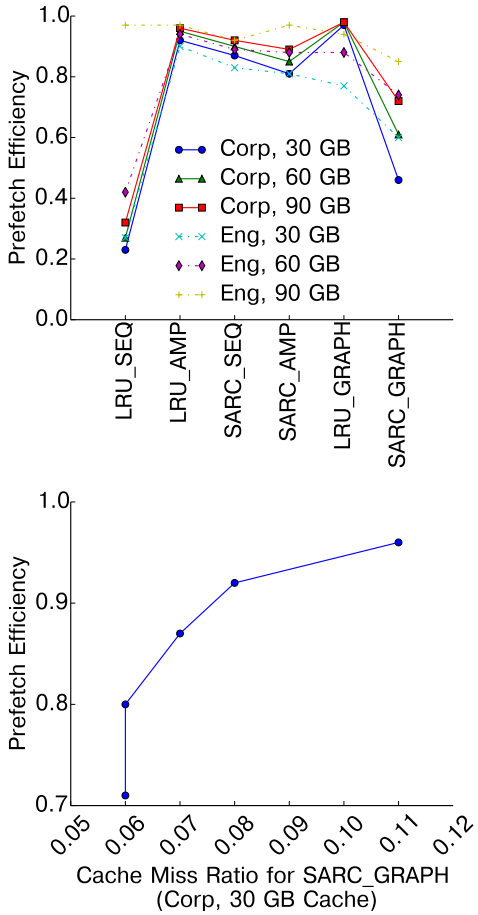

Figure 11: **Prefetch Efficiency**.

Second, we observe that *cloud gateways can mask cloud latencies even for cache misses*. Our results show that when coupled with high-performance prefetch algorithms, cloud gateways can reduce 95th percentile latency by more than 70%, and 99th percentile by nearly 40%, even though our cache hit ratio is less than 95%. This effect is more pronounced with slower cloud backends and longer cloud latencies, suggesting that our gateway could tolerate even worse cloud latencies. Traditionally, people have been using cache hit/miss ratio as the only performance indicator of caching/prefetching algorithms. However, because of high cloud latencies, partial cache misses are equally important (or arguably more important) in reducing tail latencies.

Overall, the results we report here are comparable to shared storage performance in local data centers (§ 3.2). With history-based prefetch, 90th percentile latency can be reduced to around 30 ms. *Using a cloud gateway is indeed a feasible solution for tier-2 workloads that have reasonable (not stringent) performance requirements*. Of course, certain restrictions apply: the workload needs to be cachable and has some access patterns. The two traces we study show that typical tier-2 workloads satisfy these requirements.

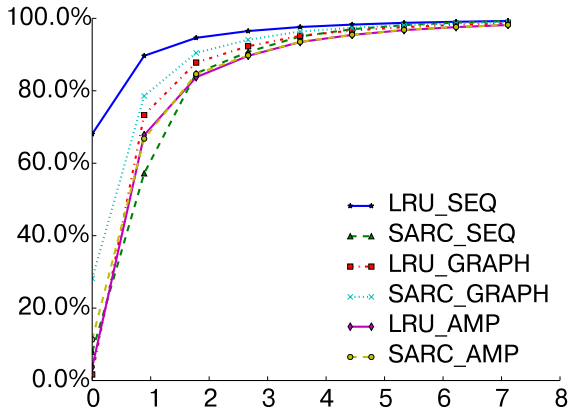

Figure 12: **Access Count for Prefetched Blocks**. Corp Dataset, 30 GB cache size.

6.3 Prefetch Efficiency and Cloud Cost

Prefetch Efficiency: One unique aspect of managing a cloud gateway stems from the monetary cost iccurred by accessing the cloud backend. Prefetch algorithms should thus try to minimize unnecessary fetches. To quantify the prefetch waste, we define a metric *prefetch efficiency* (PE):

$$PE = C_a/C_p \tag{3}$$

where C_a is the number of blocks prefetched *and* accessed and C_p is the number of blocks prefetched. Figure 11 (up) shows the prefetch efficiencies of different algorithms.

Prefetch algorithms can make tradeoffs between performance (fetching more blocks to minimize cache misses) and cost (fetching less blocks to avoid wasteful prefetch). In Figure 11 (down) we show that when varying the track size (the unit for prefetch), SARC_GRAPH may achieve lower cache-miss ratio at the cost of lower prefetch efficiency. We believe that this tradeoff is an important characteristic of any prefetch algorithm for cloud gateways, and should be thoroughly studied.

Prefetch efficiency is only an average, however. To get more detailed data, we plot the CDF of the number of times each prefetched block is accessed before eviction (Figure 12). We can see from the CDF that for most algorithms, more than 60% of the prefetched blocks are accessed only once before eviction. This may suggest that we should evict a prefetched block as soon as it is accessed to improve algorithms that have lower prefetch efficiency.

Cloud Cost: We now consider the dollar cost of cloud gateways. We divide cloud cost into two parts: the storage cost and the operational cost. Storage cost comes from storing data in the cloud, and does not change across algorithms. Operational cost comes from transferring data in and out of cloud, and the requests cost (GET/PUT/DELETE etc.). We calculate the cost per GB in a 3-years time period, and use Amazon's pricing model [8] given its popularity.

Storage cost depends on the initial data size and how data grows. For the Corporate dataset,the initial data size (DS) is 3000 GB, and the data growth rate (GR) is 58.77 GB/month (82.27 GB growth over 42 days). So after three years, the final data size is:

$$\begin{aligned} DS_{3yrs} &= DS_{init} + GR * 36 \\ &= 3000 + 58.77 * 36 = 5115.72GB \end{aligned} \tag{4}$$

and the total storage cost is:

$$\begin{aligned} Cost_{3yrs} &= (DS_{init} * 36 + GR * \sum_{i=1}^{36} i) * UnitPrice \\ &= (3000 * 36 + 58.77 * \sum_{i=1}^{36} i) * 0.00295 \\ &= \$4340.65 \end{aligned} \tag{5}$$

So the storage cost per GB for the Corporate dataset is

$$\begin{aligned} CostPerGB &= Cost_{3yrs}/DS_{3yrs} \\ &= 4040.65/5115.72 = \$0.848/GB \end{aligned} \tag{6}$$

Using a similar process we compute that the storage cost for the Engineering dataset is \$1.017/GB. It is higher than the Corporate dataset because it has a larger initial data size (19 TB) and slower data growth rate (50.33 GB/month): in general, cloud storage is cheaper for fast-growing data.

The operational cost of cloud gateways with different storage backends, algorithms and cache settings is shown in Figure 13. As mentioned previously, prefetch efficiency has an impact on operational cost, and we can see that Figure 13 does show a resemblance with Figure 11 (up).

For the Corporate dataset, the operational cost ranges from \$0.05/GB (S3) to \$0.15/GB (CloudFront) over 3 years when using history-based prefetch; for the Engineering dataset, the cost ranges from \$0.01/GB to \$0.02/GB (the cost is amortized over the larger data size). A larger local cache natually reduces cost, especially for algorithms with relatively low prefetch efficiencies, and in general operational cost is small compared to the storage cost.

	Storage	Operational	Total Cost
Corp	\$0.841	\$0.096	\$0.937
Eng	\$1.017	\$0.013	\$1.030

Table 6: **Cloud Cost per GB (3 years)**. For SARC_GRAPH, 90 GB cache.

Table 6 shows the overall cost per GB of data (after deduplication) of the cloud gateway when using SARC_GRAPH, a 90 GB cache, and the CloudFront backend (the best-performance configuration so far). To put the numbers in context, local storage applicances optimized for capacity usually cost \$1 - \$2 per GB of capacity (instead of the actual data size). For example, EMC's VNX511560010FN storage array currently costs \$1.71/GB [2]. Given the advantages of cloud storage (ease of management, availability, disaster recovery, etc.), cloud gateways are economically attractive.

6.4 Metadata Size

The metadata used to manage local caching and prefetching differ by algorithms: for LRU_AMP, it includes the

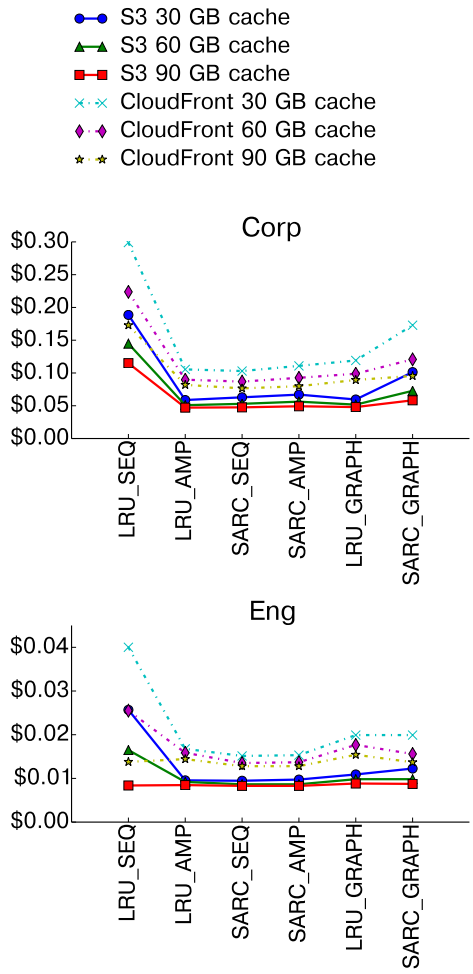

Figure 13: **Cloud Operational Cost**. Reported as $/GB over 3 years.

Algorithm/Cache size	10%	20%	30%
LRU_SEQ	2.4 GB	2.9 GB	3.4 GB
LRU_AMP	2.4 GB	2.9 GB	3.4 GB
SARC_SEQ	2.8 GB	3.8 GB	4.7 GB
SARC_AMP	2.8 GB	3.8 GB	4.7 GB
LRU_GRAPH	6.3 GB	6.8 GB	7.1 GB
SARC_GRAPH	4.3 GB	5.1 GB	6.0 GB

Table 7: **Metadata Size, Corp dataset**.

LRU list for block eviction and the stream information for prefetch; for SARC_AMP, there is additional per track information and sequentiality information; for graph based algorithms, the history graph is the most significant metadata. The on-disk metadata sizes of different algorithms for the Corp dataset are shown in Table 7. The Eng dataset shows very similar data. For both datasets, the metadata used by GRAPH prefetching algorithms is roughly 1.3-2.5 times that of LRU-based schemes. On the other hand, SARC_AMP uses about 17%-35% more metadata than LRU_SEQ.

We also observe that in GRAPH algorithms, the number of nodes in the access graph is typically only 1% of the total number of blocks, confirming that using block ranges is an effective way to reduce graph size.

7. Related Work

The cloud gateway market is nascent and several startup companies are working in this space. As mentioned previously, the most common use of cloud gateways is for backup or archival; Amazon Storage Gateway [6], Nasuni [38], Riverbed [43], TwinStrata [50] are popular commercial ones. A few cloud gateway vendors such as Panzura [41] and StorSimple [48] provide all-in-one solutions. Panzura enables teams to share data across different sites by providing cloud gateways at each site that share a single globally-deduplicated file system. StorSimple has SSD tiers with inline, variable-length deduplication. As an academic effort, BlueSky [51] is closest to cloud gateways, providing a network file system backed by cloud storage. However, it assumes too optimistic cloud latencies (12 - 30 ms), which is unlikely based on our measurements. In addition, it only deploys a naive (synchronous, full segment) prefetching strategy and does not evaluate agaist realistic enterprise workloads. There are also related works focused on other aspects of gateway design. For example, DepSky [13] provides a "cloud of clouds" model to replicate across different providers. Similarly, ViewBox [52] studies cloud-based synchronization to avoid data corruption and inconsistency.

As mentioned in Section 4, caching and prefetching techniques have been studied extensively in prior work, both at the file level [9, 10, 26, 27, 31] and at the block level [19, 24, 25, 33, 34]. We investigate these techniques and adapt them to the cloud gateway environment.

BORG [14] constructs a process access graph at the block level in a manner very similar to our system, where vertices represent block ranges and edges correlation between two block ranges. However, BORG uses the graph to organize data on local disk instead of making prefetching decisions.

Chen et. al compared the cost of moving computation and storage to the cloud, versus keep them local in 2011 [17]. They claim that it is not economically attractive to store data remotely because of the data transfer cost. However they did not consider how effective caching reduces data transferred. With the lower cloud cost today and advanced caching techniques, the picture is clearly changing.

8. Conclusion

Cloud storage gateways have been widely used for backup data. Our work shows that tier-2 primary workloads that have reasonable performance requirements can also be moved to the cloud with cloud gateways. When equipped with the right caching and prefetching techniques, cloud gateways can overcome cloud latencies and deliver good performance at low cost. In order to achieve high performance for tier-2 workloads, a cloud gateway must capture random access patterns and optimize for partial cache misses.

Acknowledgement

Feedbacks from the anonymous reviewers and Dean Hildebrand (our shepherd) have significantly improved this work. We thank the members of the NetApp ATG group and ADSL research group for their suggestions and comments.

This material was supported by funding from NSF grants CNS-1421033, CNS-1319405, CNS-1218405, CNS-1419199 as well as generous donations from EMC, Facebook, Google, Huawei, Microsoft, NetApp, Seagate, Samsung, Veritas, and VMware. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and may not reflect the views of NSF or other institutions.

References

[1] Bufferbloat. http://www.bufferbloat.net/.

[2] EMC VNX Pricing, Cost and Price List. http://www.storagepricing.org/emc-vnx-pricing-cost-and-price-list/.

[3] Linux Souce Code, fs/cifs/transport.c. https://github.com/torvalds/linux/releases/.

[4] RITE:Reducing Internet Transport Latency. http://riteproject.eu/.

[5] AllegroGraph. Allegrograph. http://www.franz.com/agraph/allegrograph/.

[6] Amazon.com, Inc. AWS Storage Gateway. https://aws.amazon.com/storagegateway/.

[7] Amazon.com, Inc. CloudFront. http://aws.amazon.com/cloudfront/.

[8] Amazon.com, Inc. Simple Storage Service. http://aws.amazon.com/s3.

[9] A. Amer, D. D. Long, and R. C. Burns. Group-based Management of Distributed File Caches. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 525–534. IEEE, 2002.

[10] A. M. Amer. *Data Grouping Using Successor Prediction*. PhD thesis, University of California, 2002.

[11] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2014.

[12] J. W. Bates. System and Method for Resource Sharing across Multi-cloud Arrays, May 5 2014. US Patent App. 14/269,758.

[13] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-clouds. In *Proceedings of the 6th ACM european conference on Computer Systems*, 2011.

[14] M. Bhadkamkar, J. Guerra, L. Useche, S. Burnett, J. Liptak, R. Rangaswami, and V. Hristidis. BORG: Block-reORGanization for Self-optimizing Storage Systems. In *Proceedings of the 7th USENIX Symposium on File and Storage Technologies (FAST '09)*, pages 183–196, San Francisco, California, Feb. 2009.

[15] Z. Bjornson. AWS S3 vs Google Cloud vs Azure: Cloud Storage Performance. http://blog.zachbjornson.com/2015/12/29/cloud-storage-performance.html.

[16] J. Brutlag. Speed Matters for Google Web Search. http://services.google.com/fh/files/blogs/google_delayexp.pdf.

[17] Y. Chen and R. Sion. To Cloud or Not to Cloud?: Musings On Costs and Viability. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, page 29, 2011.

[18] CTERA. Ctera. http://www.ctera.com/.

[19] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch. In *USENIX Annual Technical Conference*, volume 7, pages 261–274, 2007.

[20] EMC, Inc. Managing Storage: Trends, Challenges, and Options. http://tinyurl.com/p6feufx.

[21] M. Garcia and R. Woundy. Network Latency Optimization, July 24 2014. US Patent App. 13/748,980.

[22] I. Gartner. Cloud-Storage Gateways: Bridge the Gap. 2011.

[23] I. Gartner. User Survey Analysis: IT Spending Priorities in Government, Worldwide, 2013. 2013.

[24] B. S. Gill, L. Angel, and D. Bathen. AMP: Adaptive Multi-stream Prefetching in a Shared Cache. In *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST '07)*, San Jose, California, Feb. 2007.

[25] B. S. Gill and D. S. Modha. SARC: Sequential Prefetching in Adaptive Replacement Cache. In *Proceedings of the USENIX Annual Technical Conference (USENIX '07)*, Santa Clara, CA, June 2007.

[26] J. Griffioen and R. Appleton. Reducing File System Latency Using a Predictive Approach. In *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1*, USTC'94, pages 13–13, Boston, Massachusetts, 1994.

[27] P. Gu, Y. Zhu, H. Jiang, and J. Wang. Nexus: A Novel Weighted-Graph-Based Prefetching Algorithm for Metadata Servers in Petabyte-Scale Storage Systems. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1. IEEE, 2006.

[28] D. Hitz, J. Lau, and M. Malcolm. File system design for an nfs file server appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, WTEC'94, pages 19–19, San Francisco, California, 1994. USENIX Association.

[29] IBM. *IBM System Storage DS8000 Series: Architecture and Implementation*. IBM Press, 2007.

[30] S. Jiang, X. Ding, Y. Xu, and K. Davis. A Prefetching Scheme Exploiting Both Data Layout and Access History on Disk. *Trans. Storage*, 9(3):10:1–10:23, Aug. 2013.

[31] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *USENIX Annual Technical Conference*, pages 275–288, 1997.

[32] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller. Measurement and Analysis of Large-Scale Network File System Workloads. In *Proceedings of the USENIX Annual Technical Conference (USENIX '08)*, Boston, Massachusetts, June 2008.

[33] M. Li, E. Varki, S. Bhatia, and A. Merchant. TaP: Table-based Prefetching for Storage Caches. In *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST '08)*, volume 8, pages 1–16, San Jose, California, Feb. 2008.

[34] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-Miner: Mining Block Correlations in Storage Systems. In *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST '04)*, pages 173–186, San Francisco, California, April 2004.

[35] J. Liddle. Amazon Found Every 100 ms of Latency Cost Them 1% of Sales. http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/.

[36] N. Martínez-Bazan, M. Á. Águila-Lorente, V. Muntés-Mulero, D. Dominguez-Sal, S. Gómez-Villamor, and J.-L. Larriba-Pey. Efficient Graph Management Based on Bitmap Indices. In *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, IDEAS '12, pages 110–119, New York, NY, USA, 2012. ACM.

[37] A. Muthitacharoen, B. Chen, and D. Mazieres. A Low-Bandwidth Network File System. In *ACM SIGOPS Operating Systems Review*, volume 35. ACM, 2001.

[38] Nasuni. Nasuni. http://www.nasuni.com/.

[39] Neo4j. Neo4j. http://www.neo4j.org.

[40] M. A. Olson, K. Bostic, and M. I. Seltzer. Berkeley db. In *Proceedings of the USENIX Annual Technical Conference (USENIX '99)*, Monterey, California, June 1999.

[41] Panzura. Panzura. http://panzura.com/.

[42] H. Perros. Computer Simulation Techniques: The Definitive Introduction. *Computer Science Department, North Carolina State University, Raleigh, North Carolina, USA*, 2003.

[43] Riverbed. Riverbed whitewater. http://www.riverbed.com.

[44] A. Sabaa, M. A. G. Gowda, and P. Kuriakose. Fibre Channel Storage Area Network to Cloud Storage Gateway, Mar. 8 2013. US Patent App. 13/791,415.

[45] E. Schurman and J. Brutlag. Performance Related Changes and Their User Impact. http://goo.gl/hAUENq.

[46] A. Singla, B. Chandrasekaran, P. Godfrey, and B. Maggs. The Internet at the Speed of Light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2014.

[47] G. Soundararajan, M. Mihailescu, and C. Amza. Context-aware Prefetching at the Storage Server. In *Proceedings of the USENIX Annual Technical Conference (USENIX '08)*, Boston, Massachusetts, June 2008.

[48] StorSimple. Storsimple. http://www.storsimple.com/.

[49] E. Symmetrix. Symmetrix 3000 and 5000 Enterprise Storage Systems Product Description Guide, 1999.

[50] TwinStrata. Twinstrata. http://www.twinstrata.com/.

[51] M. Vrable, S. Savage, and G. M. Voelker. Bluesky: A cloud-backed file system for the enterprise. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies (FAST '12)*, San Jose, California, Feb. 2012.

[52] Y. Zhang, C. Dragga, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. ViewBox: Integrating Local File Systems with Cloud Storage Services. In *Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST '14)*, San Jose, California, Feb. 2014.

[53] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. PriorityMeister: Tail Latency QoS for Shared Networked Storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.