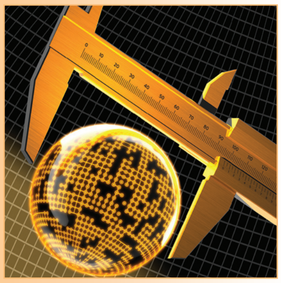

The Impact of Content Sharing on Cloud Storage Bandwidth Consumption

Content sharing in cloud storage leads to multiple downloads of the same content when users synchronize devices. These downloads contribute to bandwidth waste and increase server workloads. Here, the authors investigate traffic generated by Dropbox and use data collected from four networks to show that a large fraction (57–70 percent) of downloads generated by Dropbox users is associated with content shared among multiple devices. They present an alternative synchronization architecture that uses caches to offload storage servers from such downloads. Their experiments show that the approach cost-effectively avoids most repetitive downloads, benefiting service providers, the network, and end users.

Glauber Gonçalves
Universidade Federal de Minas Gerais, Brazil

Idilio Drago
Politecnico di Torino, Italy

Ana Paula Couto da Silva
Universidade Federal de Minas Gerais, Brazil

Alex Borges Vieira
Universidade Federal de Juiz de Fora, Brazil

Jussara M. Almeida
Universidade Federal de Minas Gerais, Brazil

Cloud storage is currently one of the most popular Internet services, generating traffic volume that has been increasing at a fast pace.[1] Indeed, the entrance of big companies (such as Google, Microsoft, and Apple) into this market confirms the lively scenario. Dropbox, a leader in the cloud storage market, has surpassed the mark of 400 million users, uploading 1.2 billion files to the Internet every 24 hours in 2015.[2]

Such services offer a practical and safe environment for both domestic and enterprise users to store and share data, facilitating content organization and collaborative work. Yet, popular features of these services – notably content sharing – pose an extra load for servers and the network, as data shared among multiple user devices might require several transfers from remote servers. This holds even if devices are close to each other (for example, within a campus network), and despite Dropbox's efforts to implement device-to-device synchronization with the LAN Sync protocol.[3] Such downloads ultimately waste network bandwidth and increase the workload at the cloud servers.

Cloud storage services employ mechanisms to reduce network traffic, such as compression and deduplication.[4] Although each of these mechanisms reduces the traffic between the cloud and user devices by up to 24 percent,[5] they don't target content sharing and, as such, have limited effect on downloads of the same content to synchronize multiple devices. This is worrisome given previous observations that downloads account for higher traffic than uploads in cloud storage.[1,6]

At the same time, Internet traffic in general presents significant redundancy[7] caused by, for example, the download of a single content by multiple users. Web caching and content delivery networks are classical solutions to offload servers and remove cross-border traffic from the network. Intuitively, these solutions could be applicable to cloud storage as well. However, most cloud storage providers don't implement any distributed synchronization architectures,[4] such as the deployment of caches nearby to end users who are connected far from datacenters. Is the traffic caused by content sharing significant for providers and, if so, are content-sharing characteristics such that caching could cost-effectively reduce its impact on servers?

In this article, we address to what extent content sharing in cloud storage leads to repetitive downloads from the cloud (and thus bandwidth waste) in current networks. This problem has only been discussed to date, though still in a preliminary manner, in our prior work.[8] Here, we perform a thorough investigation of this issue. We characterize content sharing among Dropbox users relying on traffic traces collected in four distinct networks: two university networks (one in South America and one in Europe) and two points of presence (PoP) of a European ISP.

We also assess the impact of content sharing on cloud storage traffic by quantifying the downloads associated with the same content shared by multiple devices within the monitored networks. We find that a large fraction (57–70 percent) of the downloads from Dropbox servers falls into this category. Moreover, a significant fraction of such traffic (up to 25 percent of Dropbox related incoming traffic in the monitored networks) is likely caused by the download of content replicas and, therefore, is avoidable.

We then explore whether the introduction of network caches would reduce the number of avoidable downloads in a cost-effective way. We propose a modification to the synchronization architecture of Dropbox, introducing caches to temporarily hold user updates. We evaluate this architecture in various setups, considering both typical scenarios based on our datasets and simulations with larger user populations and/or content sharing.

We show that even a reasonably small cache (for example, 70 Gbytes) could offload servers from handling almost all avoidable downloads. Even more, our proposed architecture is cost-effective: taking the traffic observed in one of our datasets, we find that 92 percent of the costs for serving avoidable downloads can be recovered after discounting the costs of the cache. Moreover, such benefits tend to increase, reaching 95–97 percent if we consider adequate cache sizes (for example, 280–500 Gbytes) and scenarios where the cache is deployed to cover larger user populations (for example, at large ISPs) and/or user populations sharing more content.

Overall, our results show that storage providers have incentives to deploy the caching-based architecture, which additionally would remove cross-border traffic from the Internet and reduce synchronization time for end users.

Content Synchronization in Dropbox

To begin, let's look at how Dropbox synchronizes content.

Basic Mechanisms

Dropbox synchronizes content by relying on two main concepts: devices and namespaces. Users can register several devices in the system. During this process, users select an initial folder from which files are synchronized with the cloud. This initial folder is visible from any other device belonging to the user. Users might share content with other users by creating shared folders, which are visible in all devices of all users participating in the sharing. Both initial folders and shared folders are the root of independent directory trees, where actual files are stored, and are called *namespaces* in the Dropbox system.[9]

We focus on the Dropbox desktop client, because it's responsible for more than 75 percent of the Dropbox traffic in 2014.[1] Devices using this client usually keep a local copy of all files present in the user's namespaces. The addition of any content in a namespace triggers content propagation: all devices having the Dropbox desktop client and registering the namespace retrieve the content immediately if online, or as soon as they come back online.

Dropbox controls the status of namespaces by means of a notification protocol, which wasn't encrypted until mid-2014. Each namespace is associated with a journal identifier (JID), representing its latest version. Devices discover when namespaces are outdated by periodically exchanging a list of namespaces and respective

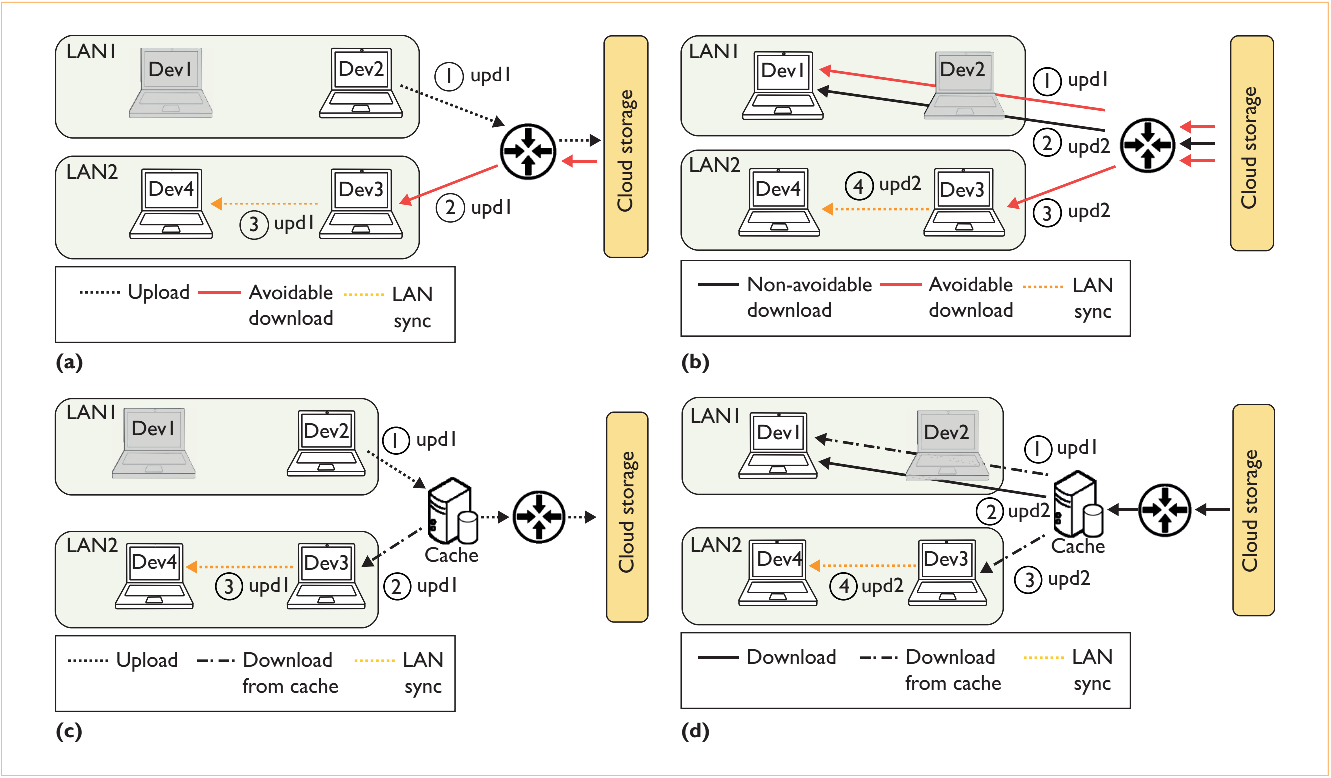

Figure 1. Content synchronization in Dropbox current architecture with common cases of avoidable downloads from the cloud to synchronize devices: update generated (a) inside the network (Dev1 is offline) and (b) outside the network (Dev2 is offline, Dev 1 is back online and retrieves upd1 and upd2 from the cloud). Our proposal for an alternative synchronization architecture: update generated (c) inside the network (Dev1 is offline) and (d) outside the network (Dev2 is offline, Dev1 is back online and retrieves upd1 and upd2 from the cache).

JIDs with Dropbox servers. If any namespace is outdated, the device executes several transactions with Dropbox servers until all namespaces become synchronized with the cloud.

We define an update as the steps that a device needs to take to move a namespace from a JID to its next JID value. Updates include files that have been added to the namespace and all metadata and commands that manipulate the namespace, such as to delete files and create folders.

By observing messages of the notification protocol, it's possible to identify when namespaces are updated. Indeed, we developed a methodology in previous work[10] to collect a long-term dataset about Dropbox namespaces, which includes the traffic volume exchanged in each JID transition of a large sample of namespaces — that is, an estimation of the update sizes.

Synchronization Architecture with LAN Sync

Dropbox deploys the LAN Sync protocol for synchronizing devices in LANs, which might prevent downloads from the cloud. The protocol works as follows: First, devices periodically broadcast information about their namespaces. Any device in the LAN can form a list of possible neighbors for future synchronizations. Next, an outdated device checks the status of neighbors before retrieving updates from the cloud. Device-to-device synchronization takes place if the namespace is already updated in any device within the LAN.

The synchronization architecture of Dropbox with LAN Sync is summarized in Figures 1a and 1b. Devices in different LANs are kept synchronized, retrieving updates either from the cloud (solid black or red arrows) or from local peers using LAN Sync (orange arrows). In practice, however, typical campus and ISP networks are subdivided into multiple LANs — that is, Dropbox broadcast messages reach a limited number of devices. Moreover, devices sharing namespaces must be online simultaneously to allow the LAN Sync protocol to work effectively.

In Figure 1, some updates (solid red arrows) need to be retrieved from the cloud to synchronize devices, even if the same updates have already been observed in the network — for example, when LAN Sync isn't effective. We call those cases *avoidable downloads*. Avoidable downloads occur either because the update has been previously uploaded by a device in the network, or because multiple devices download a single update generated elsewhere.

Characterizing Avoidable Downloads

Next, we closely analyze data we captured, to better understand the effects and dimensions of downloads that are avoidable.

Datasets and Methodology

We rely on data captured and prepared in our previous work, where we modeled the workload of cloud storage.[10] Different from that work, here we analyze content sharing and assess its impact on storage traffic. We collected the datasets by monitoring Dropbox traffic at four vantage points. Campus-1 and Campus-2 are distinct campus networks in South America and Europe. Campus-1 has a user population of ≈57,000 people, whereas Campus-2 serves ≈15,000 people. PoP-1 and PoP-2 monitor customers at PoPs of a European ISP, aggregating ≈25,000 and ≈5,000 households, respectively.

Our data includes flow-level information containing the volume exchanged by clients with Dropbox servers, and metadata extracted from Dropbox notification messages. We collected the latter by means of deep packet inspection and include, for each notification, the device ID and the list of namespaces with respective device JIDs. Note that client IP addresses are anonymized and notifications offer no hints about users' identities. In total, we observed around 23 Tbytes of Dropbox traffic, 27,428 unique Dropbox devices, and 61,419 unique namespaces.

Dropbox clients exchange notifications with servers once a minute when online. When a namespace is updated, additional Dropbox traffic appears in the network, and a notification message announcing the new JID is sent out by the client. We thus process traces (see also our previous work[10]) to estimate each update's size, correlating flow volumes with notification messages, as follows:

- flows and notifications are first grouped per client IP address;
- flows and notifications overlapping in time form a synchronization cycle;
- synchronization cycles are classified as upload, download, or mixed, based on downstream and upstream volumes; and
- update sizes are estimated for clean cycles, while the remaining cycles are discarded.

We consider a synchronization cycle to be clean when a single device is active — either uploading or downloading — given an IP address, or if multiple devices are active sharing an IP address, but a single namespace is changed. We assume the latter to be a typical network address translator (NAT) scenario, where one device uploads content that's spread to peer devices. We know which devices are active even behind NATs, and the likely uploader, thanks to the device IDs found in notification messages. Among situations that prevent us from estimating update sizes, we highlight multiple devices editing various namespaces simultaneously behind a NAT; and the synchronization happening when devices reappear online, which often involves many namespaces and both uploads and downloads. Finally, the volume in a clean cycle is divided equally among updates in the cycle, for simplicity.

Overall, we retain more than 63 percent of the Dropbox traffic for our analysis (see Table 1). We use the resulting dataset to study avoidable downloads. We track all updates of namespaces and mark downloads as avoidable when a namespace is updated to a specific JID in a device, and at least one other device registering the namespace has announced the same JID. Note that our methodology doesn't take into account synchronizations performed using LAN Sync, because such traffic doesn't reach our probes. We thus identify avoidable downloads after LAN Sync actuates.

Results

Table 1 lists results, including the number of namespaces shared by at least two devices in the monitored networks (that is, shared namespaces); traffic volumes retained for the analysis; traffic volumes associated with shared namespaces, and avoidable downloads; and datasets' duration.

Shared namespaces. Table 1 shows that the percentage of shared namespaces in campuses (≥33 percent) is higher than in PoPs (≤26 percent). The number of distinct devices registering each shared namespace is, however, small. More than 90 percent of the shared namespaces

Table 1. Uploads and downloads in Dropbox (note the shared and avoidable volumes).

Dataset	Shared namespace	Retained volume* (Gbytes)			Shared namespaces volume** (Gbytes)			Months
		Upload	Download	Total	Upload	Download	Avoidable	
Campus-1	3,787 (36%)	1,304 (77%)	2,282 (60%)	3,586 (65%)	605 (46%)	1,425 (62%)	411 (18%)	3
Campus-2	3,254 (33%)	326 (53%)	443 (44%)	769 (47%)	129 (40%)	253 (57%)	74 (17%)	3
PoP-1	7,514 (26%)	2,604 (67%)	3,909 (57%)	6,513 (61%)	1,482 (57%)	2,601 (67%)	761 (19%)	7
PoP-2	2,925 (24%)	1,850 (77%)	2,558 (68%)	4,408 (72%)	1,077 (58%)	1,801 (70%)	637 (25%)	11
Total	17,480 (28%)	6,084 (71%)	9,192 (60%)	15,276 (64%)	3,293 (54%)	6,080 (66%)	1,883 (20%)	–

*Percentages refer to the dataset before discarding updates. Differences are compatible with the prevalence of network address translators (NATs) in the networks.

**Percentages refer to the sample in which we can estimate update sizes.

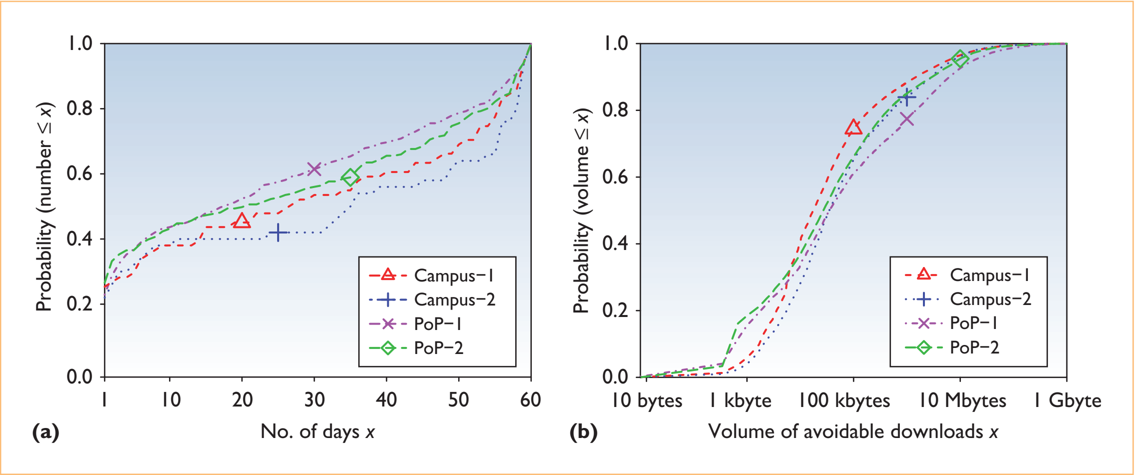

Figure 2. Shared namespaces lifespan and volume of avoidable downloads per namespace update. (a) Cumulative distribution functions of shared namespace lifespans. (b) Cumulative distribution functions of traffic volume associated with avoidable downloads (log scale on the x-axis).

are registered by two devices only. The fraction of namespaces shared by at least three devices is somewhat higher in the campus datasets, reaching 7–9 percent. In the PoP datasets, this fraction falls to the 4–6 percent range. Such a difference is unsurprising and has been associated with the use of Dropbox for collaborative work in campuses, and the synchronization of a user's different devices at home.[10]

We find that users' interest on namespaces tends to last for a short time: after an initial period, the number of accesses (that is, updates and downloads) in a namespace becomes negligible. We illustrate this behavior in Figure 2a by showing cumulative distribution functions of the number of days between the first and last access in the first 60 days of life of namespaces. We refer to this time period as the *namespace lifespan*. Moreover, we focus on namespaces created during our captures and consider only namespaces seen online for at least 60 days.

Notice how all accesses occur on the day the namespace first appears for 22–27 percent of the namespaces. The median lifespan of a namespace is around one month. There is, however, a non-negligible number of namespaces with long lifespans: for example, around 20 percent of the namespaces in PoP-1 still present activity 50 days after the first access. Yet, overall, we find that shared namespaces tend to have short lifespans — that is, their accesses occur with a strong temporal locality, which seems to favor the deployment of network caches.

Avoidable traffic. Table 1 shows a high volume of downloads in all datasets. We observe that 44–68 percent of the traffic corresponds to downloads, and 57–70 percent of such download traffic belongs to shared namespaces. This raises a question about how much traffic is associated with content replication. Not all downloads are avoidable, because devices might retrieve content uploaded in other networks, for example, when a user has remote devices, or namespaces shared with users located somewhere else. We see, however, that the percentage of avoidable downloads is quite significant. Overall, we find that up to 25 percent of the Dropbox download traffic happens to synchronize content that has been observed in the same network.

Figure 2b presents the total download volume that's avoidable per update. Note that most updates that cause such downloads generate little traffic. Yet, a non-negligible portion results in large volumes. For example, while 25 percent of the updates in Campus-1 generate at least 100 Kbytes of replication, more than 3 percent of the updates surpass 10 Mbytes, reaching up to 1 Gbyte of replication.

Summary. Our analysis confirms that downloads dominate Dropbox traffic, and a significant part of such downloads is avoidable, contributing to increased costs for the provider. Table 1 shows significant percentages of avoidable downloads in four networks, each with a different prevalence of NATs, indicating that results aren't influenced by the data discarded by our methodology to estimate the size of updates.

Shared namespaces have a limited lifespan. Updates on those namespaces are typically small and present strong temporal locality. These results motivate us to investigate next whether the introduction of network caches could reduce the number of avoidable downloads in a cost-effective way.

A New Synchronization Architecture

We propose a new synchronization architecture for cloud storage that consists of introducing network caches to temporally hold user updates in the network. We aim to enable device synchronization, without the need to retrieve content from the cloud in scenarios where the LAN Sync protocol is ineffective. Figures 1c and 1d illustrates our proposal in the same cases shown in Figures 1a and 1b. A storage cache is installed to cover several networks and possibly thousands of customers — for example, multiple LANs in a large campus, complete ISP networks, or multiple PoPs. As a consequence, devices can potentially find updates without retrieving content from the cloud.

Synchronizing devices using the storage caches works as follows. The discovery of caches is orchestrated by Dropbox protocols when devices log in to the system. Dropbox servers inform clients about the closest cache in the network. Devices always send updates in namespaces to the closest cache (see Figure 1c, step 1). The cache stores updates locally, also forwarding them to the cloud. As soon as devices receive notifications about updates, they contact the closest cache. If the pending updates exist in the cache (that is, a cache hit), the cache delivers the content directly to devices. Otherwise (that is, a cache miss), the cache retrieves the content from the cloud and forwards it to the requesting devices (see Figure 1d, steps 2 and 3). After any request from a client, the cache executes internal replacement policies to guarantee that the most likely useful content is available locally. Any caching replacement policy could be employed for that matter (see the book by Michael Rabinovich and Oliver Spatschek for references[11]), and an evaluation of the best caching policy is outside the present scope.

We envision the caches being deployed and controlled by cloud storage providers directly. Moreover, we don't assume any particular deployment topology. The location of caches in the network would be a choice of operators, and the only requirement is that clients must have routes to reach the caches. The extensions needed in Dropbox protocols to diverge traffic to caches, as well as other aspects related to the architecture's implementation, are also out of our scope. Finally, we evaluate next whether the caching approach is cost-effective for storage providers, considering the cache costs and resulting bandwidth savings. Equally important questions, such as the privacy risks and the management overhead of deploying caches in several locations, are left for future work.

Evaluation Methodology

We evaluate the architecture using both our datasets and synthetic traces. We first use real traces to study whether storage caches are cost-effective in typical campus and ISP networks. Then, we rely on synthetic traces to extrapolate measurements and understand how costs and benefits vary when large populations are covered, or more sharing is seen in the network.

For the synthetic traces, we create an environment as in Figure 1. A number of devices act independently, performing updates that trigger several downloads. A network cache is deployed to cover the devices. It intercepts updates, potentially serving content without involving the cloud. For simplicity, we assume all content is uploaded from the simulated environment — that is, external devices produce no workload to be downloaded.

We rely on CloudGen for creating synthetic traces. CloudGen is a synthetic workload generator for cloud storage services presented in our other work.[10] CloudGen allows us to realistically reproduce the traffic created by arbitrary populations of Dropbox devices, based on parameters learned from traffic traces (for example, devices' sessions duration). CloudGen generates a synthetic trace, indicating when each device is online, the updates in each namespace, and the data volume associated with each update.

We generate four types of synthetic traces:

- *Typical workload*. We set CloudGen to the number of devices observed in our datasets to illustrate how the synthetic workload compares to real traces.
- *High population*. We simulate large populations by tripling the number of devices — that is, roughly equivalent to having a Dropbox client in every IP address observed in Campus-1. Both upload and download volumes grow linearly with the number of devices.
- *High sharing*. We triple the mean number of devices sharing each namespace. The scenario mimics environments where users share lots of namespaces — for example, hypothetical companies adopting cloud storage as network file systems. Larger numbers of devices per namespace increase only download traffic.
- *High population and sharing*. We combine the previous two scenarios. Uploads grow linearly with the number of devices, while downloads grow as much as 20 times when compared to the typical scenario.

We calculate savings considering different cache sizes for both real and synthetic traces. We use the first month in each dataset of real traces to warm up the cache, and assess savings in subsequent months. For synthetic scenarios, we generate two-month-long intervals (warm up and evaluation) in 16 experiment rounds, and report mean values with 95 percent confidence intervals.

In all experiments, we use the simple and yet popular least-recently-used (LRU) policy: cache insertions and evictions are triggered by uploads; downloads manipulate the order of items (that is, updates) in the LRU cache, but don't change cache contents. Finally, by tracking updates and cache behavior, we identify whether an update should be retrieved from the cache or cloud.

Performance of the Cache-Based Architecture

Now that we've detailed the architecture, let's look at its performance.

Bandwidth savings. First, let's consider the bandwidth savings obtained with our cache-based architecture, focusing on Campus-1's real trace, as well as on several synthetic traces generated using CloudGen (parameterized according to that dataset). We omit results for the other datasets for the sake of brevity. Yet, conclusions in the following hold for all of the datasets.

We evaluate bandwidth savings using the byte hit ratio metric, which is computed as the volume of downloads served by the cache over the total volume of downloads reaching the clients. Figure 3a reports the byte hit ratios for two months extracted from the real trace (dashed purple curves) as well as a synthetic trace in the typical scenario (continuous black curve).

The differences between the two months in real traces are due to normal variations in the workload: month 3 presents higher volume in shared namespaces than month 2, which reduces the byte hit ratio, as we discuss next. Despite some divergences, the synthetic traces capture reasonably well the overall trend in real data. Such divergences are due to multiple factors. First, the synthetic trace is built by configuring CloudGen with input parameters extracted from the complete Campus-1 trace, as opposed to a subtrace corresponding to a particular month. Thus, it captures an overall trend observed during the whole three-month period, and not the particular behavior in shorter periods. Moreover, some simplifying assumptions made in the design of CloudGen contribute to the divergences, particularly for small caches. For instance, CloudGen assumes that devices behave independently, which might not be always the case. Multiple devices sharing namespaces might be online simultaneously, favoring the caching approach.

Yet, despite such divergences, we still see overall common trends. For example, for both

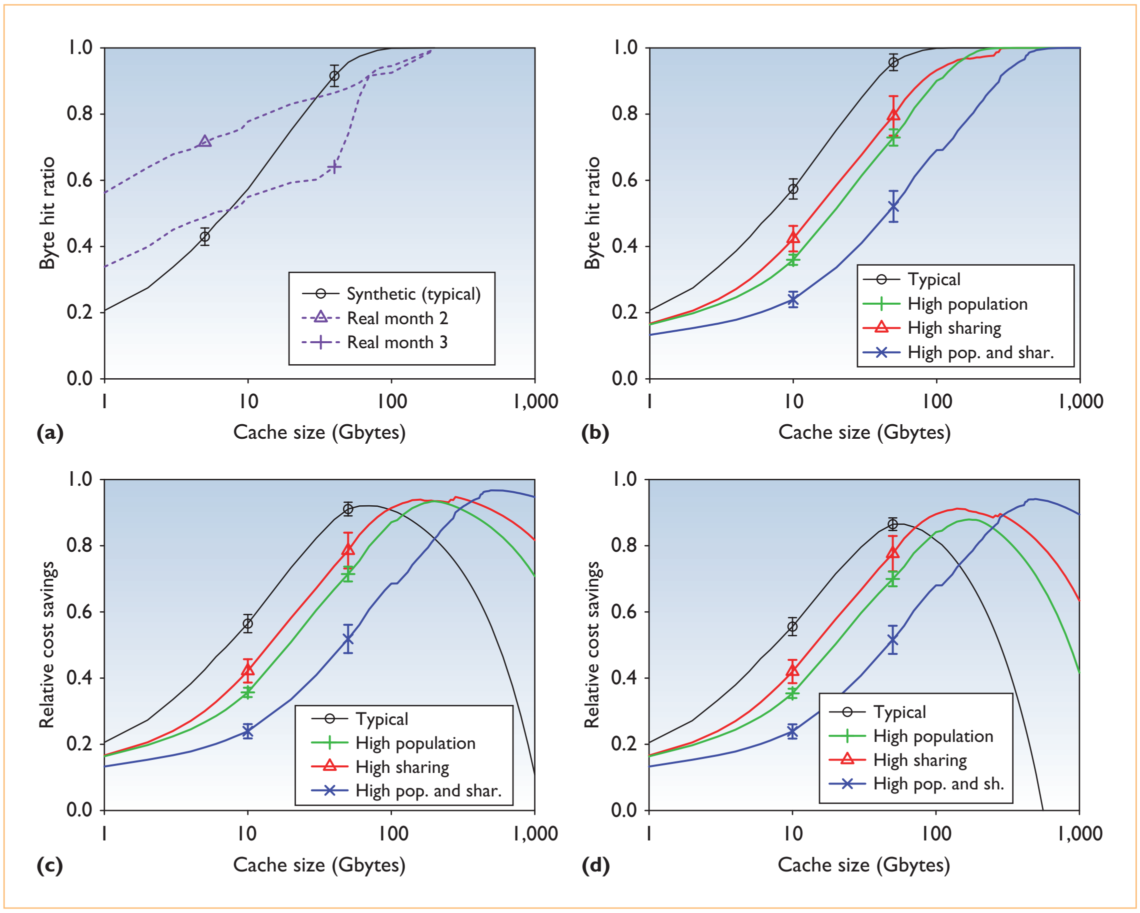

Figure 3. Performance of the cache-based architecture in various scenarios. (a) Real and synthetic. (b) Synthetic scenarios. (c) Storage prices are half of the bandwidth prices (β/α = 1/2). (d) Storage prices are equal to bandwidth prices (β/α = 1).

real and synthetic workloads, even a small cache is able to remove a large fraction of avoidable downloads. In fact, a cache of 1 Gbyte would achieve from 21–56 percent of the byte hit ratio. Moreover, in all three curves, the byte hit ratio surpasses 90 percent with a 70-Gbyte cache.

Having compared real and synthetic traces in the typical scenario, we focus on the latter to evaluate the proposed architecture in other scenarios. Results are in Figure 3b. We note that, for any given cache size, the byte hit ratio is higher in the typical scenario than in any of the other three scenarios. This is because workloads are heavier in other scenarios, resulting in more cache misses. When the population size is increased, the larger upload volumes force the cache to remove old updates more often. In the high sharing scenario, the larger number of devices downloading content (that is, sharing namespaces) increases the probability that a device will fail to find a particular content in the cache, because the content has already faced eviction. Cache savings decrease even more in the high population and sharing scenario, as both factors are present. Nevertheless, despite such variations, our proposed architecture provides significant bandwidth savings in all scenarios. For example, for a 10-Gbyte cache, the byte hit ratio varies from 24–57 percent.

Overall, we conclude that even a modest cache can remove most of the Dropbox avoidable downloads when deployed in a typical network. As larger scenarios are considered, the cache size to achieve similar savings needs to be adjusted.

Cost-benefit tradeoff. The tradeoffs of deploying our architecture involve the cache costs and savings achieved by removing downloads from the cloud. In other words, an effective cache should prevent avoidable downloads at a minimum cost.

We evaluate costs and benefits of the architecture by computing the relative cost savings (*rcs*) for a time interval *t*:

$$rcs_t = \frac{\text{cost\_nocache}_t - \text{cost\_cache}_{t,c}}{\text{cost\_nocache}_t},$$

where cost_nocache_t is the cost to serve all avoidable downloads in the time period *t*, while $\text{cost\_cache}_{t,c}$ corresponds to the cost associated with deploying a cache of size *c* bytes during time period *t*.

In short, rcs_t captures the fraction of the costs to serve avoidable downloads that the architecture can recover. The architecture is effective when $rcs_t > 0$; that is, the savings obtained by removing avoidable downloads at least pay off the costs associated with maintaining the cache. Note that rcs_t has an upper bound ($rcs_t = 1$) that indicates a scenario where all avoidable downloads are removed at negligible costs.

For the sake of simplicity, we express the cost_nocache and the cost_cache as functions of the number of bytes transmitted over the network and stored in the cache:

$$\text{cost\_nocache}_t = d_t * \alpha$$
$$\text{cost\_cache}_{t,c} = m_{t,c} * \alpha + c * \beta,$$

where d_t is the total number of bytes associated with avoidable downloads observed in the period *t*, $m_{t,c}$ is the number of bytes associated with avoidable downloads that a cache of size *c* bytes misses during time *t*, and α and β represent bandwidth and storage prices per byte, respectively.

We take as reference for α and β the prices for bandwidth and storage offered by Amazon Simple Storage Service and Elastic Compute Cloud (S3/E2C),[12] which already include operational costs. Based on that, we evaluate rcs_t on two distinct cost setups, defined by the ratio β/α: the price per byte of storage is a half of the bandwidth (that is, $\beta/\alpha = 1/2$), for example, because magnetic storage is used; and both prices are the same (that is, $\beta/\alpha = 1$), for example, SSD storage is taken. We use Amazon S3/E2C as reference because many storage providers rely on it to build their services. Other references (for example, market prices for SSD disks and mobile data plans) provide more optimistic cost-benefit tradeoffs.

Figures 3c and 3d show the rcs_t for the four scenarios and the two price ratios considering *t* equal to one month. The value of rcs_t increases to a maximum and then decreases. The inflection occurs when the best tradeoff between costs and benefits is achieved. Compared to Figure 3b, we see that the decrease in rcs_t happens despite higher byte hit ratios. Therefore, caches larger than a certain threshold add costs to the system, without providing significant savings.

Figure 3c ($\beta/\alpha = 1/2$) shows that rcs_t reaches 92–93 percent for typical and high population scenarios with cache sizes of 70 and 200 Gbytes, respectively. Scenarios with higher sharing achieve slightly better rcs_t. The maximum rcs_t reaches 95 percent for a 280-Gbyte cache in the high sharing scenario, going up to 97 percent in the high population and sharing scenario with a 500-Gbyte cache. Better tradeoffs with higher sharing are explained by the large volume of downloads: even if the byte hit ratio is lower than in typical scenarios for certain cache sizes (see Figure 3b for 100-Gbyte caches), the volume saved with avoidable downloads pays back the investment.

It's worth noting the effects of the ratio between storage and bandwidth prices: the higher the storage price (compared to bandwidth), the lower the advantage of deploying the architecture. Comparing Figures 3c and 3d, we see that the maximum rcs_t slightly decreases for $\beta/\alpha = 1$; for example, it's 7 percent lower than for $\beta/\alpha = 1/2$ in the typical scenario. Moreover, the costs of overestimated caches become equivalent to the bandwidth savings faster when storage costs are high.

In sum, our proposal cost-effectively achieves high bandwidth savings in a typical scenario, where 92 percent of the costs relative to avoidable downloads can be recovered. Storage providers thus have an incentive to deploy the caches, because savings by far compensate the costs. Benefits are higher with high-sharing (rcs_t reaches up to 95–97 percent), which hints to networks where cloud storage providers should start deploying the architecture.

We showed that content sharing in cloud storage services causes an expressive volume of avoidable traffic to providers. We proposed the use of network caches for content synchronization aiming at reducing this traffic. Our analyses suggest that even a small cache yields very high bandwidth savings. Such savings potentially cover the costs associated with building up the system, thus calling on cloud storage providers to consider the deployment of the architecture as a competitive advantage to their services. Future directions include analyzing privacy risks and the management overhead of deploying storage caches in different locations. This will require new models and simulations to support providers' decisions on where to deploy caches.

Acknowledgments

This research is partially funded by the FAPEMIG-PRONEX-MASWeb project — Models, Algorithms, and Systems for the Web, process number APQ-01400-14; by the National Institute of Science and Technology for the Web, CNPq, and FAPEMIG; and by the Vienna Science and Technology Fund through project ICT15-129, BigDAMA.

References

1. E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage: Usage, Performance, and Impact of Terminals," *Proc. 4th IEEE Int'l Conf. Cloud Networking*, 2015, pp. 106–111.
2. Dropbox, "What Is Dropbox?" 2015; www.dropbox.com/news/company-info.
3. M. Dee, *Inside LAN Sync*, blog, 13 Oct. 2015; https://blogs.dropbox.com/tech/2015/10/inside-lan-sync.
4. E. Bocchi, I. Drago, and M. Mellia, "Personal Cloud Storage Benchmarks and Comparison," to be published in *IEEE Trans. Cloud Computing*; doi:10.1109/TCC.2015.2427191.
5. Z. Li et al., "Towards Network-Level Efficiency for Cloud Storage Services," *Proc. ACM Internet Measurement Conf.*, 2014, pp. 115–128.
6. R. Gracia-Tinedo et al., "Dissecting UbuntuOne: Autopsy of a Global-Scale Personal Cloud Back-End," *Proc. ACM Internet Measurement Conf.*, 2015, pp. 155–168.
7. A. Anand et al., "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination," *Sigcomm Computer Comm. Rev.*, vol. 38, no. 4, 2008, pp. 219–230.
8. G. Gonçalves et al., "Analyzing the Impact of Dropbox Content Sharing on an Academic Network," *O Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* [Proc. Brazilian Symp. Computer Networks and Distributed Systems], 2015 (in Portuguese), pp. 100–109.
9. N. Koorapati, *Streaming File Synchronization*, blog, 2014; https://blogs.dropbox.com/tech/2014/07/streaming-file-synchronization.
10. G. Gonçalves et al., "Workload Models and Performance Evaluation of Cloud Storage Services," to be published in *Computer Networks*, 2016; doi:10.1016/j.comnet.2016.03.024.
11. M. Rabinovich and O. Spatschek, *Web Caching and Replication*, Addison-Wesley Longman Publishing, 2002.
12. Amazon.com, "Amazon EC2 Pricing," 2015; https://aws.amazon.com/ec2/pricing.

Glauber Gonçalves is pursuing a PhD in computer science from Universidade Federal de Minas Gerais, Brazil. His research interests include computer networks, distributed systems, and performance modeling. Gonçalves has an MS in computer science from Universidade Federal de Minas Gerais, Brazil. Contact him at ggoncalves@dcc.ufmg.br.

Idilio Drago is a postdoctoral researcher at the Politecnico di Torino, Italy, in the Department of Electronics and Telecommunications. His research interests include Internet measurements, Big Data analysis, and network security. Drago has a PhD in computer science from the University of Twente. He was awarded an Applied Networking Research Prize in 2013 by the IETF/IRTF for his work on cloud storage traffic analysis. Contact him at idilio.drago@polito.it.

Ana Paula Couto da Silva is an associate professor at the Universidade Federal de Minas Gerais, Brazil. Her research interests include modeling and analysis of computer systems and energy efficient networking, characterization of Internet traffic, and complex network theory. Da Silva has a PhD in computer and system engineering from the Federal University of Rio de Janeiro. Contact her at ana.coutosilva@dcc.ufmg.br.

Alex Borges Vieira is an associate professor at the computer science department of Universidade Federal de Juiz de Fora, Brazil. His research interests include P2P, dynamic networks, and wireless sensor networks, with an emphasis on system modeling and performance analysis. Vieira has a PhD in computer science from Universidade Federal de Minas Gerais. Contact him at alex.borges@ufjf.edu.br.

Jussara M. Almeida is an associate professor of computer science at the Universidade Federal de Minas Gerais, Brazil. Her research interests include performance modeling of large-scale distributed systems. Almeida has a PhD in computer science from the University of Wisconsin–Madison. Contact her at jussara@dcc.ufmg.br.