Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective

Binbing Hou
Louisiana State University
bhou@csc.lsu.edu

Feng Chen
Louisiana State University
fchen@csc.lsu.edu

Zhonghong Ou
Beijing Univ. of Posts & Telecomm.
zhonghong.ou@bupt.edu.cn

Ren Wang
Intel Labs
ren.wang@intel.com

Michael Mesnier
Intel Labs
michael.mesnier@intel.com

***Abstract*—Cloud storage has gained increasing popularity in the past few years. In cloud storage, data is stored in the service provider's data centers, and users access data via the network. For such a new storage model, our prior wisdom about conventional storage may not remain valid nor applicable to the emerging cloud storage. In this paper, we present a comprehensive study and attempt to gain insight into the unique characteristics of cloud storage, primarily from the client's perspective. Through extensive experiments and quantitative analysis, we have acquired several interesting, and in some cases unexpected, findings. (1) Parallelizing I/Os and increasing request sizes are keys to improving the performance, but optimal bandwidth may only be achieved with a proper combination of parallelism and request size. (2) Client capabilities, including CPU, memory, and storage, play an unexpectedly important role in determining the achievable performance. (3) A geographically long distance affects client-perceived performance but does not always result in lower bandwidth and longer latency. Based on our experimental studies, we further present a case study on appropriate chunking and parallelization in a cloud storage client. Our studies show that specific attention should be paid to fully exploiting the capabilities of clients and the great potential of cloud storage services.**

***Index Terms*—Cloud Storage; Storage Systems; Performance Analysis; Measurement.**

I. Introduction

Cloud storage is a quickly growing market. According to a report from Information Handling Services (IHS), personal cloud storage subscriptions increased to 500 million in 2012 and will reach 1.3 billion by 2017 [37]. The global market is expected to grow from $18.87 billion in 2015 to $65.41 billion by 2020 [5]. To date, cloud storage is not only used for archiving personal data, but also plays an indispensable role in various core commercial services, from serving videos on demand to storing unstructured scientific data.

To end users, cloud storage is particularly interesting because it provides a compelling new storage model. In this model, data is stored in the service provider's data centers, and users access data through an HTTP-based REST protocol via the Internet. By physically and logically separating data storage from data consumers, this architecture enables enormous flexibility and elasticity, as well as the highly desirable cross-platform capability. On the other hand, such a model is drastically distinct from conventional direct-attached storage – the "storage medium" is replaced by a large-scale storage cluster, which may consist of thousands of massively parallelized machines; the "I/O bus" is the worldwide Internet, which allows connecting two geographically distant ends; the strictly defined "I/O protocol" is replaced by an HTTP-based protocol; the "host" is not a single computing entity any more but could be any kind of computing devices (e.g., PCs or Smartphones). All these properties, together, form a rather loosely coupled system, which is fundamentally different from its conventional counterpart. A direct impact of such change is that much of our prior wisdom about storage, the basis for our system optimizations, may not continue to be applicable to the emerging cloud-based storage.

This is because of several reasons. First, the massively parallelized storage cluster, where data is stored, potentially allows a large amount of independent parallel I/Os to be processed quickly and efficiently. In contrast, our conventional storage emphasizes how to organize sequential I/O patterns to address the limitation of rotating mediums [25], [39]. Second, compared to the stable and speedy I/O bus, such as the Small Computer System Interface (SCSI), the lengthy Internet connection between the client and the cloud is slow, unstable, and sometimes unreliable. A cloud I/O could travel an excessively long distance (e.g., thousands of miles from coast to coast) to the service provider's data center, which may involve dozens of network components and finally result in an I/O latency of hundreds of milliseconds or even more. Finally, the clients, which consume the data and drive the I/O activities, are highly diverse in all aspects, from CPU, memory, storage, to communication. Certain specifications (e.g., CPU) are directly related to the capability of a client for handling parallel network I/Os.

Unfortunately, our current understanding on storage behaviors, are mostly confined in the conventional storage, which is well-defined and heavily tuned to scale in a limited scope, such as direct attached storage or local Storage Area Network (SAN). Without a thorough and detailed study, it is difficult to obtain key insights on the unique I/O behaviors of cloud storage, a storage solution for cloud stacks, especially from the perspective of data consumers. In this paper, we attempt to answer a set of important questions listed as follows.

978-1-4673-9055-2/16/$31.00 © 2016 IEEE

Successfully answering these questions cannot only help us understand the effect of several conventional key factors (e.g., parallelism and request sizes) on cloud I/O behaviors, but also several new issues (e.g., client capabilities, geo-distances), which are unique to the cloud-based storage model.

- Parallelization and request size are two key factors affecting the performance of storage. What are their effects on the performance of cloud storage? Can we make a proper tradeoff between parallelism degree and request size?
- CPU, memory, and storage are three major components defining the capability of a client. In the scenario of cloud storage, which is the most critical one affecting the performance of cloud storage? What are their effects on the performance under different workloads?
- The geographical distance between the client and the cloud determines the Round Trip Time (RTT), which is assumed to be a critical factor affecting the cloud storage speed. What is the effect of such geographical distance to cloud I/O bandwidths and latencies? Should we always attempt to find a nearby data center of a cloud storage provider?
- Based on our experimental studies on the performance of cloud storage, what are the associated system implications? How can we use them to optimize client applications to efficiently exploit the advantages of cloud storage?

In this paper, we present a comprehensive experimental study on cloud storage and strive to answer these critical questions. Unlike some prior studies that primarily focus on the cloud storage providers (e.g., [27], [28], [29]), we pay specific attention to the client side. In essence, our study regards cloud storage as a type of storage service rather than network service. As such, we are more interested in characterizing the end-to-end performance perceived by the client, rather than the intermediate communications. We believe this approach also echoes the demand for thoroughly understanding cloud storage for a full-system integration as a storage solution [22].

For our experiments, we develop and run a homemade test tool over Amazon Simple Storage Services (S3). By using latencies and bandwidths, which are the two key metrics used in storage studies, we perform a series of experiments with five different client settings to study the effect of clients' capabilities and locations. Based on our experimental studies, we further study several optimization issues on the client side, such as identifying a proper chunk size for caching and parallelization for prefetching. We hope this work can provide a complete picture of cloud storage and inspire the research community, especially cloud storage system and application designers, to further leverage the unique characteristics of cloud storage for effective optimizations.

The rest of the paper is organized as follows. Section II introduces background. Section III describes the methodology for our experimental studies. Section IV and V present the results and case study. Section VI discusses the system implications of our findings. Related work is presented in Section VII and the last section concludes this paper.

II. Background

A. Cloud Storage Model

In cloud storage, the basic entity of user data is an *object*. An object is conceptually similar to a file in file systems. An object is associated with certain metadata in the form of key/value pairs. Typically, an object can be specified by a URL consisting of a service address, bucket, and object name (e.g., https://1.1.1.1:8080/v1/AUTH_test/c1/foo). The maximum object size is typically 5GB, which is the limit of the HTTP protocol [15]. Objects are further organized into logical groups, called *buckets* or *containers*. A bucket/container is akin to a directory in a file system but cannot be nested.

Almost all cloud storage service providers offer an HTTP-based Representational State Transfer (REST) interface to users for accessing cloud storage objects. Some also provide language-specific APIs for programming. Two typical operations are `PUT` (uploading) and `GET` (downloading), which are akin to `write` and `read` in conventional storage. Other operations, such as `DELETE`, `HEAD`, and `POST`, are provided to remove objects, retrieve and change metadata. For each operation, a URL specifies the target object in the cloud storage. Additional HTTP headers may be attached as well.

B. Cloud Storage Services

Cloud storage is designed to offer convenient storage services with high elasticity, reliability, availability, and security guarantees. Amazon S3 [3] is one of the most typical and popular cloud storage services. Other cloud storage services, such as OpenStack Swift [10], share a similar structure. Typically, the cloud storage service is running on a large-scale storage cluster consisting of many servers for different purposes, from handling HTTP requests, accounting, storage, to bucket listings, etc. These servers could be further logically organized into *partitions* or *zones* based on physical locations, machines/cabinets, network connectivity and so on. For reliability, the zones/partitions are isolated with each other, and data replicas should reside separately. In short, the cloud storage services are built on a massively parallelized structure and are highly optimized for throughput.

C. Cloud Storage Applications

Applications can access cloud storage in different ways. Some applications use the vendor-provided APIs to directly program data accesses to the cloud in their software. Such APIs are provided by the service provider and are usually language specific (e.g., Java or Python). Since a cloud storage object can be located via a specified URL, users can also manually generate HTTP requests by using tools like `curl` to access the link.

A more popular category of cloud storage applications is for personal file sharing and backup (e.g., Dropbox). Such applications often provide a filesystem-like interface to allow end users to access cloud storage. From the perspective of data exchange, these clients often use syncing or caching to enhance user experience. With the syncing approach, the client

Client	Instance	Location	Zone	vCPU	Memory	Storage	Network	OS
Baseline	m1.large	Oregon	us-west-2a	2	7.5GB	Magnetic(410GB)	Moderate	Ubuntu 14.04 (PV)
CPU-plus	c3.xlarge	Oregon	us-west-2a	4	7.5GB	Magnetic(410GB)	Moderate	Ubuntu 14.04 (PV)
MEM-minus	m1.large	Oregon	us-west-2a	2	3.5GB	Magnetic(410GB)	Moderate	Ubuntu 14.04 (PV)
STOR-ssd	m1.large	Oregon	us-west-2a	2	7.5GB	SSD(410GB)	Moderate	Ubuntu 14.04 (PV)
GEO-Sydney	m1.large	Sydney	ap-southeast-2a	2	7.5GB	Magnetic(410GB)	Moderate	Ubuntu 14.04 (PV)

TABLE I
CONFIGURATIONS OF AMAZON EC2-BASED CLIENTS. THE SSD IS THE PROVISIONED SSD WITH 3,000 IOPS.

Speed / Size	Magnetic		SSD	
	Read	Write	Read	Write
1KB	2.13 MB/s	0.77 MB/s	2.7 MB/s	1.24 MB/s
4KB	6.70 MB/s	3.13 MB/s	10.57 MB/s	5.67 MB/s
16KB	6.80 MB/s	4.60 MB/s	34.87 MB/s	10.65 MB/s
64KB	7.36 MB/s	10.67 MB/s	62.00 MB/s	28.48 MB/s
256KB	17.36 MB/s	17.46 MB/s	58.24 MB/s	86.63 MB/s
1MB	38.33 MB/s	22.38 MB/s	58.24 MB/s	82.71 MB/s
4MB	61.59 MB/s	23.20 MB/s	58.06 MB/s	82.72 MB/s
16MB	58.12 MB/s	22.66 MB/s	58.12 MB/s	82.92 MB/s

TABLE II
MAGNETIC VS. SSD

Object Size	Object Number	Workload Size
1KB	81920	80MB
4KB	40960	160MB
16KB	40960	640MB
64KB	40960	2560MB
256KB	40960	10240MB
1MB	16384	16384MB
4MB	4096	16384MB
16MB	2048	32768MB

TABLE III
OBJECT-BASED WORKLOADS

maintains a complete copy of the data stored on the cloud-side repository. A syncer daemon monitors the changes and periodically synchronizes the data between the client and the cloud. With the caching approach, the client only maintains the most frequently used data in local, and any cache miss leads to on-demand data fetching from the cloud. In practice, the syncing mode is adopted by almost all personal cloud storage applications, such as Dropbox [6], Google Drive [8], OneDrive [9], etc. The caching mode is adopted by the applications and storage systems that make use of the cloud as a part of the I/O stack, such as RFS [26], S3FS [13], S3backer [12], BlueSky [45], SCFS [18], etc. In general, all the above-mentioned applications essentially convert the POSIX-like file operations into an HTTP-based protocol to communicate with the cloud. For the sake of generality, our study carefully avoids using any specific application techniques but uses the raw HTTP requests.

III. MEASUREMENT METHODOLOGY

As mentioned above, the main purpose of our experimental studies is to characterize the performance behaviors of cloud storage from the client's perspective. In our experiments, we treat the cloud as a "blackbox" storage. In order to avoid interference from client-side optimizations, we carefully generate raw cloud I/O traffic via the HTTP-based REST protocol to directly access the cloud storage and observe the performance on the client side.

Cloud storage services: Our experiments are conducted on Amazon Simple Storage Services (S3). As a representative cloud storage service, Amazon S3 is widely adopted as the basic storage layer in consumer and commercial services (e.g., Netflix and EC2). Some third-party cloud storage services, such as Dropbox, are directly built on S3 [7]. In our experiments, we use the S3 storage system hosted in Amazon's data center in Oregon (s3-us-west-2.amazonaws.com).

Cloud storage clients: In order to run the experiments in a stable and well-contained system, we choose Amazon EC2 as our client platform from which the cloud storage I/O traffic is generated to exercise the target S3 repository. An important reason of choosing Amazon EC2 rather than our own machines is to have a quantitatively standardized client that provides a publicly available baseline for repeatable and meaningful measurement. For analyzing the impact of client variance, we customized five configurations of Amazon EC2 instances which feature different capabilities in terms of CPU, memory, storage, and geographical location. Table I shows these configurations. The *Baseline* client is located in Oregon and equipped with 2 processors, 7.5 GB memory, and 410 GB disk storage (denoted as Magnetic). The speeds of the Magnetic and the SSD are tested and shown in Table II. The other four configurations vary in different aspects, specifically CPU, memory, storage, and geographical location (in Sydney). These instances can properly satisfy our needs of observing cloud storage performance with differing clients.

Test workloads: For our experiments, we develop a home-made tool by using the S3 API [11] to generate raw cloud I/O requests to S3. We purposely avoid using POSIX APIs (e.g., S3FS) because our goal is to gain the direct view of the cloud storage performance from the client side. Certain techniques (e.g., local cache, data deduplication, data compression) used in some client tools will prevent us from observing the cloud I/O behaviors completely or accurately. Our tool allows us to create combinations of various parallelism degrees (1-64), object sizes (1KB to 16MB), and types (PUT or GET). Before each run, we generate objects of the same size with unique keys/names in the client storage and upload to the cloud as the uploading workloads; we then download the objects to the client. Table III lists more details about the workloads.

Accuracy: Considering the possible variance of network services and multi-thread scheduling, we take the following

measures to ensure the accuracy and repeatability of the experiments: (1) As stated above, we customize the instances of Amazon EC2 which can provide stable services as standard clients rather than picking up a random machine. (2) To avoid memory interferences across experiments, the memory is flushed before each run of the experiments. (3) We make the size of the workloads large enough (see Table III) so that each run of an individual experiment lasts for a sufficiently long duration (at least 60 seconds) while still being able to complete the experiments within a reasonable time frame. (4) Each experiment is repeated for five times, and we report the average value while discarding obvious outliers.

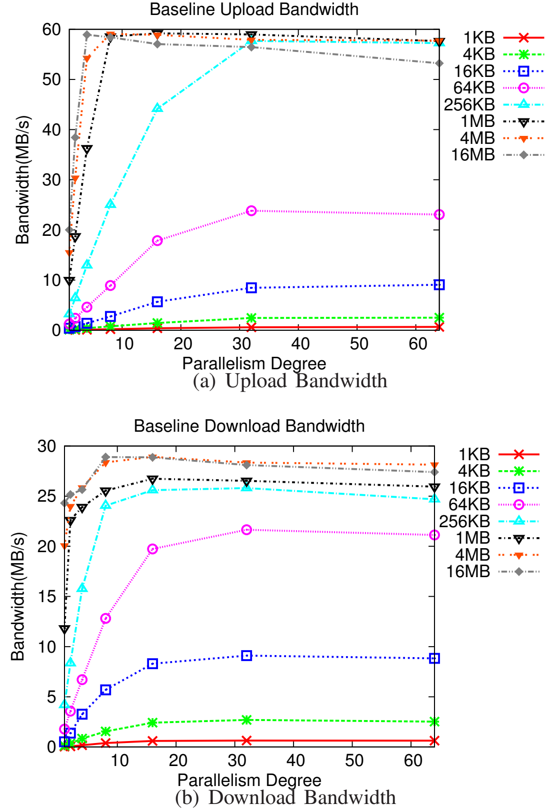


Fig. 1. Bandwidth on Baseline

IV. Performance Studies

To comprehensively reveal the effects of different factors, our measurement work is composed of two parts. We first conduct a set of general experiments to evaluate the properties of cloud storage, including parallelism degree and request size. We then focus on studying the effects of client capabilities, including CPU, memory, storage, and geographical locations of the clients.

A. Basic Observations

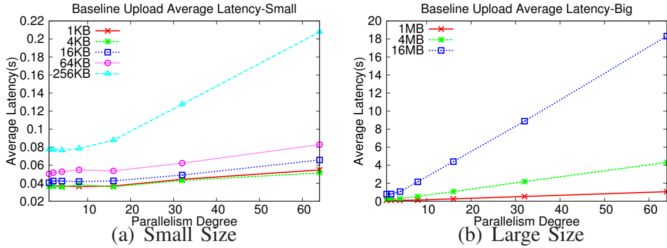
Parallelism and request size are two critical factors that significantly affect the storage performance. Considering the parallelism potential of cloud storage, we set the parallelism degree up to 64. With regard to request size, prior work has found that most user requests are not excessively large [19], typically smaller than 10MB [29]. Also, for transfer over the Internet, most cloud storage clients split large requests into smaller ones. Wuala and Dropbox, for example, adopt 4MB chunks, and Google Drive uses 8MB chunks, while OneDrive uses 4MB for upload and 1MB for download [19]. Therefore, we set the request size up to 16MB to study the size effect.

1) The effect of parallelism: In conventional disk drives, I/O parallelism has limited effect due to its mechanic nature. For cloud storage, which stores data in a cluster of massively parallelized storage servers, I/O parallelism has a significant impact to the client-perceivable performance.

Q1: How does parallelism affect the bandwidth?
Generally, proper parallelization can dramatically improve the bandwidth, while over-parallelization may lead to bandwidth degradation to certain degree. As shown in Figure 1, for example, the bandwidth of 1KB upload requests can be improved up to 27-fold (from 0.025MB/s to 0.666MB/s), and the bandwidth of 1KB download requests can be improved up to 21-fold (from 0.03MB/s to 0.634MB/s). There are two reasons for this. One reason is due to the underlying TCP/IP protocol for communication. With TCP/IP, the client and the cloud have to send ACK messages to confirm the success of the transmission of data packets. With a high parallelism degree, multiple flows can continuously transmit data since the time taken by each parallel request to wait for the ACK messages overlaps. Another reason is that smaller requests often require fewer client resources, so the client can support a higher parallelism degree to saturate the pipeline until the effect of parallelization is limited by one of the major client resources.

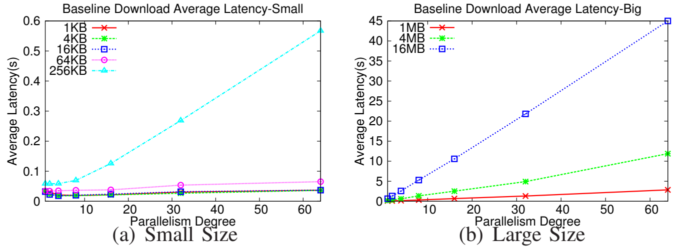
Over-parallelization brings diminishing benefits and even negative effects. For example, 16MB upload sees a slight performance degradation caused by over-parallelization. This is related to the overhead of maintaining the thread pool when the CPU is overloaded. To confirm this, we use `vmstat` in Linux to investigate the CPU utilization of the 16MB upload request with different parallelism degrees. As shown in Figure 4, when the parallelism degree is 8, the CPU utilization quickly grows close to 100%, indicating that the CPU is overloaded. Under this condition, further increasing the parallelism degree will only increase the overhead of maintaining the thread pool, which will consequently reduce the overall performance. In a later section, we will further study the effect of CPU.

Q2: How does parallelism affect the latency?
In general, proper parallelization does not affect the latency (i.e., end-to-end request completion time) significantly, while over-parallelization leads to a substantial increase of latency. As shown in Figure 2 and Figure 3, this speculation is confirmed by the tendencies of the growing average latencies for both upload and download requests as the parallelism degree increases. For example, for 4KB upload requests, when the parallelism degree increases from 1 to 16, the average latency basically remains the same (about 36ms). When the parallelism degree further increases from 16 to 64, the average latency increases by 43% (from 36.1ms to 51.5ms). For large requests, when the parallelism degree exceeds a threshold, the average



(a) Small Size (b) Large Size

Fig. 2. Average Upload Latency on Baseline



(a) Small Size (b) Large Size

Fig. 3. Average Download Latency on Baseline

latency increases linearly. For example, for 16MB upload requests, when the parallelism degree increases from 4 to 64 (16-fold), the average latency increases from 1.1s to 18.3s (17.3-fold). This implies that for latency-sensitive applications, over-parallelizing large requests should be carefully avoided.

2) The effect of request size: In conventional storage, request size is crucial to organizing large and sequential I/Os and is important in amortizing the disk head seek overhead. A similar effect has also been observed in cloud storage.

Q1: How does request size affect the bandwidth?

As expected, increasing request size (i.e., the size of GET/PUT) can significantly improve bandwidth, but the achieved benefit diminishes as request size exceeds a threshold. As shown in Figure 1(a) and Figure 1(b), the peak bandwidths of large requests and small requests have a significant gap. For example, the peak bandwidth of 4MB upload requests is 23.5 times that of 4KB upload requests (58.9MB/s vs. 2.5MB/s); the peak bandwidth of 4MB download requests is 10.7 times that of 4KB download requests (28.9MB/s vs. 2.7MB/s). There are two reasons for this phenomenon. One reason is that larger I/O requests on client storage generally have higher I/O speeds than small ones. As shown in Table II, the 4MB read speed is 9.2 times that of 4KB (61.6MB/s vs. 6.7MB/s), while the 4MB write speed is 7.4 times that of 4KB (23.2MB/s vs. 3.1MB/s). The other reason is that larger requests have higher efficiency of data transmission via network due to the packet-level parallelism [36].

Similar to parallelization, increasing the request size cannot bring an unlimited bandwidth increase, due to the constraint of other factors. For example, the speed of client storage is limited. Uploaded objects need to be first read from the local device, and downloaded objects need to be written to the local device. As shown in Table II, when the request size grows from 4MB to 16MB, the speed of Magnetic improves slightly, which limits the I/O speed of the client side. Also, the maximum size of the TCP window is limited, though tunable [4], [38]. When the request size exceeds a certain threshold, the benefit brought by increasing the request size diminishes. Our observations have confirmed this speculation. In the scenario of a single thread, as shown in Figure 5, when the request size increases from 16MB to 64MB, the upload bandwidth increases only slightly (20MB/s vs. 21.9MB/s), and the download bandwidth remains the same (24.3MB/s). In addition, other factors, such as the link bandwidth on the route, processing speed on the cloud side, etc., can also limit the achievable bandwidth. All these observations demonstrate that the benefit obtained by increasing request size is significant but is not unlimited.

Q2: How does request size affect the latency?

In general, both larger requests and highly parallelized small requests have longer latencies. For example, as shown in Figure 2 and Figure 3, when the parallelism degree is 1, the average latency of 4MB download requests is 192ms – 5.8 times that of 1MB download requests (33ms). However, when taking parallelism degree into consideration, things become different. For example, the average latency of 4MB download requests at parallelism degree 1 is 192ms, which is 13.8 times lower than the average latency of 1MB download requests at parallelism degree 64 (2.9s). Therefore, *without considering the latency increase caused by over parallelization, it is not safe to say larger requests imply longer latencies*.

As expected, for small requests, even at the same parallelism degree, the latencies do not necessarily increase as the request size increases. Figure 2(a) shows that the average latencies of 1KB and 4KB upload requests are nearly the same. Similarly, in Figure 3(a), we find that the average latencies of 1KB, 4KB, and 16KB download requests are nearly equal. The request latency is mainly composed of three parts: data transmission time via network, client I/O time, and other processing time. For small requests, the data transmission time only accounts for a small portion of the overall latency, while the other two dominant parts remain mostly unchanged, which makes the latencies of small requests similar. Also, since the maximum TCP window is 64KB by default, considering the parallelism of network [36], the transmission time of the data that are smaller than 64KB is supposed to be similar.

3) Parallelism vs. Request size: In prior sections, we find that either increasing the parallelism degree or increasing the request size can effectively improve the bandwidth, but both of them have limitations. Here naturally comes an interesting question: does there exist a combination of parallelism degree and request size to achieve the optimal bandwidth?

Answering this question has a practical value. Consider the following case: if we have a 4MB object to upload, we can choose to upload it by a single thread or split it into four 1MB chunks and upload the them in parallel. Which is faster? Figure 6 shows the performance under different combinations of parallelism degree and request size. Obviously, 256KB×16 has the highest bandwidth (44.2MB/s), which is about 3 times of the lowest (14.5MB/s). This shows that a proper

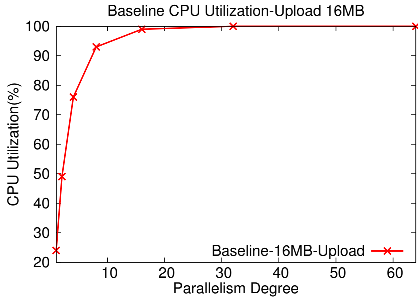
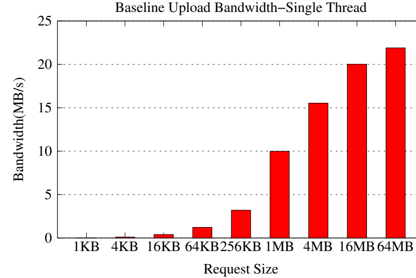
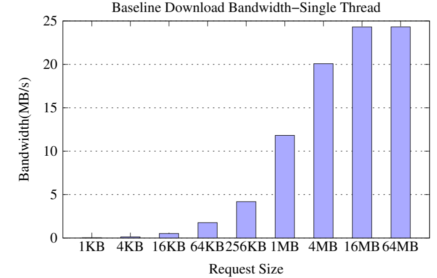


Fig. 4. Baseline CPU Utilization



(a) Upload Bandwidth



(b) Download Bandwidth

Fig. 5. Baseline Bandwidth-Single Thread

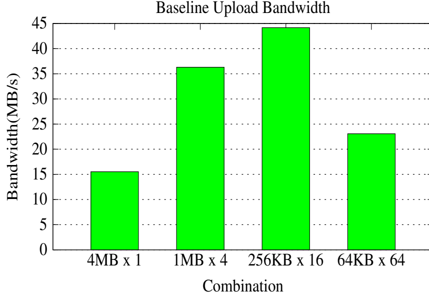
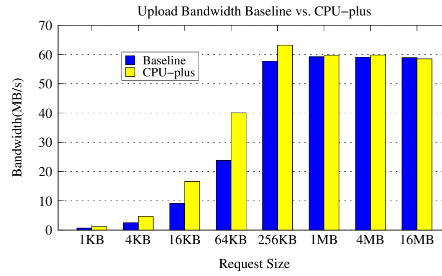
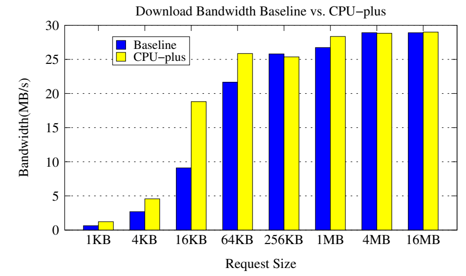


Fig. 6. Request Combinations on Baseline



(a) Upload Bandwidth



(b) Download Bandwidth

Fig. 7. Peak Bandwidth (Baseline vs. CPU-plus)

combination exists and can achieve optimal performance. This observation confirms that *appropriately combing request size and parallelism degree can sufficiently improve the bandwidth beyond optimizing only one dimension*.

We also find that, *in some cases, either increasing parallelism degree or increasing request size by the same factor can achieve the same bandwidth improvement*. For example, for upload requests, 1KB$\times$16, 4KB$\times$4, and 16KB$\times$1 have similar bandwidth (0.4MB/s). Here comes another practical question: if we have a set of small files (e.g. 1KB), should we adopt a high parallelism degree (e.g. 16) or bundle the small files to achieve large request size (e.g. 16KB)? From the perspective of improving bandwidth, either high parallelism degree or large request size is feasible. However, from the perspective of the utilization of client resources, we find that a large request size requires less CPU resources. Through `vmstat` in Linux, we find that the CPU utilization of the above three cases are 65%, 15% and 5%, respectively. This indicates that for the combinations that can achieve comparable bandwidth, a larger request size consumes less CPU resources. That is because for a larger request size, fewer threads have to be maintained to achieve the similar bandwidth, which consequently reduces the CPU utilization.

B. Effects of Client Capabilities

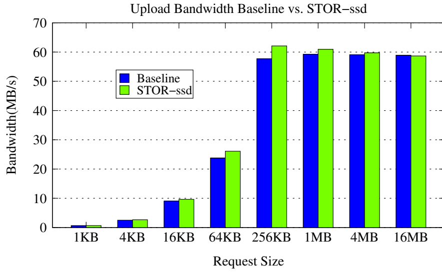
Unlike conventional storage, cloud storage clients are very diverse. In this section, we study different factors affecting the client's capabilities of handling cloud storage I/Os, namely CPU, memory, and storage. We compare the performance of three different clients, including *CPU-plus*, *STOR-ssd* and *MEM-minus*, with the performance of the *Baseline* to reveal the effects of each factor.

1) The effect of the client CPU: In cloud storage I/Os, the client CPU is responsible for both sending/receiving data packets and client I/O. In this section, we try to investigate the effect of client CPU by comparing the performance of Baseline (2 CPUs) and CPU-plus (4 CPUs).

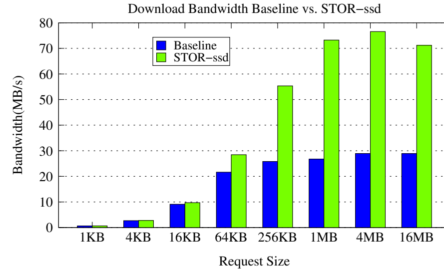
Q1: What is the effect of client CPU on bandwidth? *The client CPU has a strong impact on cloud I/O bandwidth, especially for small requests*. Figure 7 shows the peak bandwidth, which is the maximum achievable bandwidth with parallelized requests. We can see that the peak bandwidth of small requests (smaller than 256KB) increases significantly. Interestingly, as shown in Figure 7(b), the peak download bandwidth of 1KB, 4KB and 16KB requests doubles, as the computation capability doubles (2 CPUs vs. 4 CPUs). This vividly demonstrates that small requests are CPU intensive, and as so, small requests receive more benefits from a better CPU.

Large requests, compared to small ones, are relatively less sensitive to CPU resources, as the system bottleneck shifts to some other components. As shown in Figure 7, compared with Baseline, the peak upload and download bandwidth of large requests (256KB to 16MB) increases only slightly. For example, the peak upload bandwidth of 4MB requests increases by 1.4% (59.2MB/s vs. 60MB/s), while the peak download bandwidth of 4MB requests is basically the same (28.9MB/s vs. 28.8MB/s). The system bottleneck may result from the limitation of other factors, such as memory or storage, rather than CPU.

Q2: What is the effect of the client CPU on latency? In our tests, we find that *the client CPU does not have significant effects on average latency*. For small requests, the data transmission via network dominates the overall latency,



(a) Upload Bandwidth



(b) Download Bandwidth

Fig. 8. Peak Bandwidth (Baseline vs. STOR-ssd)

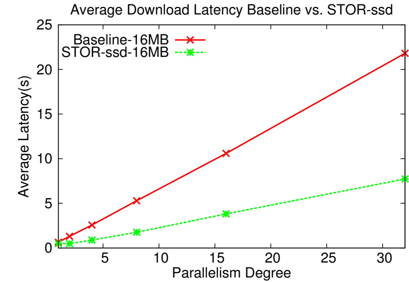


Fig. 9. 16MB Average Download Latency (Baseline vs. STOR-ssd)

while for large requests, the majority of the overall latency is the client I/O time (the I/O waiting time may be significant when client storage becomes the bottleneck) and the cloud response time, In these two cases, a more powerful CPU does not help reduce the latency.

2) The effect of client storage: Client storage plays an important role in data uploading and downloading: For uploading, the data is first read from the client storage; for downloading, the data is written to the client storage. To evaluate the effect of client storage, we set up a comparison client STOR-ssd. The only difference between Baseline and STOR-ssd is storage (Magnetic vs. SSD). Table II shows more details about the two client storage.

Q1: What is the effect of the client storage on bandwidth? We find that *client storage is a critical factor affecting the achievable peak bandwidth*. As shown in Figure 8, on STOR-ssd, the peak download bandwidth increases significantly. For example, the peak download bandwidth of 4MB requests increases by 165% (76.6MB/s vs. 28.9MB/s). On the other hand, we also notice that the upload bandwidth increases slightly. Different from the significant improvement of download bandwidth, for example, the peak upload bandwidth of 4MB requests increases only by 2% (60.3MB/s vs. 59.2MB/s). The reason why STOR-ssd improves the upload bandwidth only slightly is that the Magnetic in our experiments can achieve a similar peak read speed as SSD with a sufficiently large request size and parallelism degree. In contrast, the download bandwidth is limited by the relatively slow speed of Magnetic on the client. We have also tested with a ramdisk on Baseline. The bandwidths can be further improved but to a limited extent (77.2MB/s for uploading and 80.3MB/s for downloading).

Q2: What is the effect of the client storage on latency? *Similar to bandwidth, we did not observe significant effects of client storage to small requests and large upload requests*. For small requests, client I/O is the minority of the overall latency. In this case, client storage is not a critical factor. For large upload requests, since Magnetic and SSD have similar read speed, the latency is comparable; however, for large download requests, STOR-ssd can substantially reduce the latency because STOR-ssd have significantly advantageous write speed. For example, as shown in Figure 9, when the parallelism degree is 1, STOR-ssd can reduce the latency by 24% (0.49s vs. 0.64s); when parallelism degree is 32, the latency can be reduced by 65% (7.7s vs. 21.8s).

3) The effect of client memory: Memory in the clients has two functions. First, memory is responsible for offering running space for parallel requests. Second, memory acts as a buffer for uploading and downloading. In this section, we shrink the memory of Baseline to investigate the performance differences. The only configuration difference between MEM-minus and Baseline is that Baseline has 7.5GB memory while MEM-minus has only 3.5GB.

	1MB	**4MB**	**16MB**
Baseline	59.2MB/s	59.1MB/s	58.9MB/s
MEM-minus	58.9MB/s	58.7MB/s	58.7MB/s

TABLE IV
PEAK UPLOAD BANDWIDTH (BASELINE VS. MEM-MINUS)

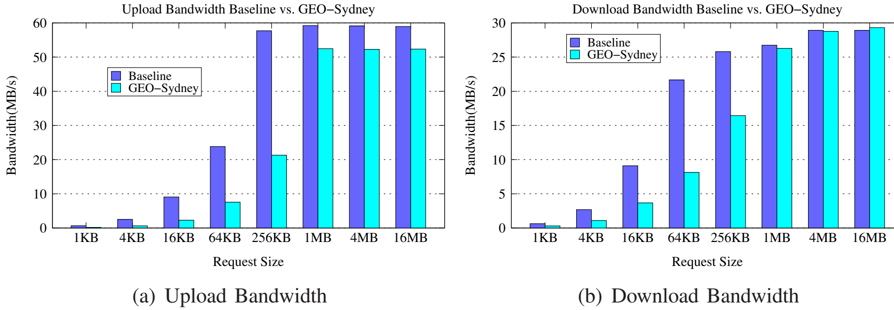
	1MB	**4MB**	**16MB**
Baseline	26.7MB/s	28.9MB/s	28.9MB/s
MEM-minus	23.7MB/s	23.8MB/s	20.8MB/s

TABLE V
PEAK DOWNLOAD BANDWIDTH (BASELINE VS. MEM-MINUS)

Since small requests are not memory intensive, the effect of memory is trivial. We only present the bandwidths of large requests. The peak upload bandwidth is basically the same (see Table IV) while the download bandwidth dropped heavily (see Table V). For example, on MEM-minus, the bandwidth of 16MB download is 20.81MB/s, which is 28.0% lower than that on Baseline (28.90MB/s). That is because the write speed of the Magnetic is much lower than read speed and thus more sensitive to the memory space. Therefore, large download requests, especially those involving intensive writes on the client, suffer more from limited memory.

C. Effects of Geographical Distance

Unlike conventional storage, for cloud storage, the geographical distance between the client and the cloud determines the Round-Trip Time (RTT), which accounts for a significant part of the observed I/O latency. The RTT between the Baseline client and the cloud is 0.28ms, as both are in the same Oregon data center. In contrast, the RTT between the GEO-Sydney client and the cloud in Oregon is about 628 times longer (176ms). This section discusses the effects of geographical distance.



(a) Upload Bandwidth

(b) Download Bandwidth

Fig. 10. Peak Bandwidth (Baseline vs. GEO-Sydney)

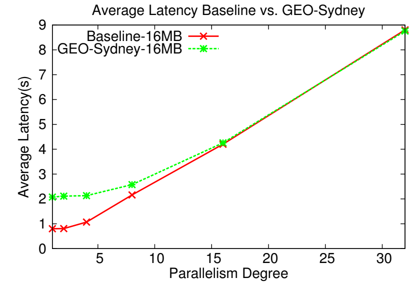


Fig. 11. 16MB Average Upload Latency (Baseline vs. GEO-Sydney)

Q1: What is the effect of geo-distance to bandwidth? *The effect of geographical distance to the achievable peak bandwidth is weaker than expected*. As shown in Figure 10, the peak upload bandwidth of GEO-Sydney is close to that of Baseline. For example, the peak upload bandwidth of 4MB requests of GEO-Sydney is only 10% lower than that of Baseline (53.3MB/s vs. 59.2MB/s) while the peak download bandwidths of 4MB download requests are basically the same (29.3MB/s vs. 28.9MB/s). This means that RTT is not a critical factor affecting the peak bandwidth, which is mostly due to the Bandwidth-Delay Product (BDP) of the network and is also consistent with the conclusion obtained by Burgen et al. [16] that the perceived bandwidth from the client is largely determined by the client's network capabilities and the network performance between the client and the cloud.

At the same time, it is also noticeable that *the achievable peak bandwidth of small requests (smaller than 1MB) is much lower with long geo-distance*. That is because a long RTT needs a high parallelism degree to saturate the pipeline of parallel requests. However, as analyzed in Section IV-B1, small requests with high parallelism are more CPU intensive; therefore, the CPU capability will be a critical bottleneck to sufficiently saturating the pipeline.

Q2: What is the effect of geo-distance to latency? As expected, we also find that *the geo-distance would significantly increase the latency, and its impact to latency makes the client less sensitive to the negative effects caused by over-parallelization to latency*. As shown in Figure 11, when the parallelism degree is 1, the average latency of 16MB upload requests on GEO-Sydney is 2.1s, which is about 2.6 times of the counterpart on Baseline (0.8s); as the parallelism degree increases, the average latencies gradually get closer; when the parallelism degree reaches 16, the average latencies are comparable (4.3s vs. 4.2s). If we compare the two, GEO-Sydney shows a flatter curve than Baseline, because a long RTT needs a high parallelism degree to saturate the pipeline, so the negative effect of over-parallelization appears later.

V. Case Study: Caching and Prefetching

In cloud storage, client-side caching and prefetching are two basic schemes for enhancing the user experience. In this section, we present a case study to show that cloud storage I/O performance could be affected by optimizing caching and prefetching. In specific, we will discuss two key techniques, *chunking* and *parallelization*.

To evaluate the effects of chunking and parallelized prefetching for cloud-based file systems, we build an emulator to implement the basic read/write operations of a typical cloud-based file system with the support of disk caching on the client. To drive this experiment, we use an object-based trace by converting a segment of an NFS trace, which is a mix of email and research workload collected at Harvard University [30]. The size of the workload in our experiments is 4.8 GB, and the average file size is 12.9 MB. For our experiments, we use Amazon S3 (in Oregon) as the cloud storage provider, and a workstation on our campus (in Louisiana) as the client. The client is equipped with a 2-core 1.2 GHZ CPU, 8GB memory, a 450GB disk drive, and installed with Ubuntu 12.04.5 LTS and Ext4 file system.

A. Proper Chunk Size for Caching

Chunking is an important technique used in cloud storage. In S3Backer, for example, the space of the cloud-based block driver is formatted with a fixed block size that can be defined by the user [12]. The choice on chunk sizes can affect caching performance: the smaller the chunk is, the less a cache miss cost would be, but the more cloud I/Os could be generated.

Although it is difficult to accurately determine the optimal chunk size, our findings about the effect of chunk size to the performance of cloud storage can guide us to roughly choose a proper, if not optimal, chunk size. We can identify a relatively small chunk size for reaching an approximately maximum bandwidth by making a reasonable tradeoff between the cache hit ratio and cache miss penalty. Figure 12 shows that, when the chunk size exceeds 4MB, the download bandwidth reaches its peak. Based on this, we speculate that the proper chunk size is possibly around 4MB. This is for two reasons. First, further increasing the chunk size over 4MB (e.g., 8MB or 16MB) cannot deliver a higher bandwidth. For example, on a cache miss of 8MB data, downloading one 8MB chunk takes an almost equal amount of time as downloading two 4MB chunks, while using 8MB chunks increases the risk of downloading irrelevant data. Second, if the chosen chunk size is excessively smaller than 4MB (e.g., 64KB or 1MB), the cache may suffer from a high cache miss ratio and cause too many I/Os.

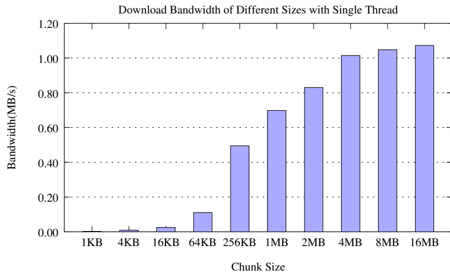


Fig. 12. Download Bandwidth of Different Sizes with Single Thread

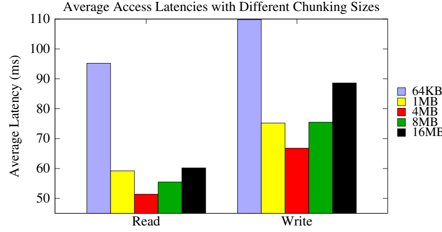


Fig. 13. Avg. Access Latencies

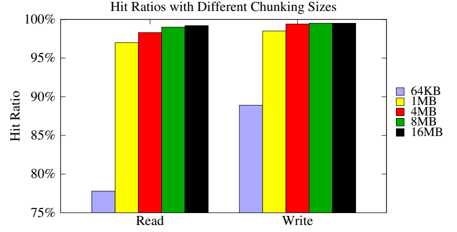


Fig. 14. Hit Ratio

To verify this speculation, we adopt the standard LRU algorithm with asynchronous writeback (for the purpose of generality). Every 30 seconds, we flush dirty data back to the cloud. The cache size is set as 200MB disk space. Besides a 4MB chunk size, for a comparison, we choose two smaller chunk sizes, 64KB and 1MB, and two larger chunk sizes, 8MB and 16MB, to study the effect of the chunk sizes.

The average access latencies with different chunk sizes are shown as Figure 13. It clearly shows that the lowest average read/write latencies are achieved at 4MB, which confirms our speculation. When the chunk size increases from 64KB to 4MB, the average read latency decreases by 47.3% (from 95.2ms to 50.2ms), and the average write latency decreases by 40.4% (from 109.9ms to 65.5ms). This benefit is due to the increase of cache hit ratio: The read hit ratio increases from 77.8% to 98.4%, and the write hit ratio increases from 88.9% to 99.4% (see Figure 14). This is mostly because using a relatively large chunk size allows to pre-load the useful data and consequently improves the cache hit ratio and the overall performance. However, when the chunk size exceeds a certain threshold, further increasing chunk size may cause undesirable negative effects. Figure 14 shows that the cache hit ratios increase slightly with a large chunk size. The increased cache miss penalty with a large chunk size is responsible for the slowdown. Specifically, it takes 4s to load a 4MB chunk, while it needs 14.2s for 16MB. Consequently, the average access latencies increase.

The analysis above has shown how to determine the proper chunk size for a certain client. Specifically, 4MB is the proper chunk size on our client for the testing workload. For the workloads with weak spatial locality, the proper chunk size should be correspondingly smaller. In general, an excessively large chunk size is not desirable, as it increases the risk of unnecessary overhead with no extra benefit.

B. Proper Parallelization for Prefetching

Prefetching is another widely used technique in cloud storage. Since objects can be downloaded (prefetched) in parallel, a proper parallelism degree is important to the performance, while over-parallelization may raise the risk of mis-prefetching and also waste resources.

In order to determine a proper parallelism degree for a certain chunk size, it is critical to ensure that on-demand fetching would not be significantly affected by prefetching. To find the proper parallelism degree that will not significantly increase the average fetching latencies, an exhaustive search on the client is feasible but inefficient. Based on our findings, in fact, we can greatly simplify the process of identifying a proper parallelism degree. To show how to achieve this, we take the chunk sizes 64KB, 1MB and 4MB as examples. We may first choose a 4MB chunk with parallelism degree 1 and then gradually increase the parallelism degree step by step (i.e., 2, 4, 8) for testing. For smaller chunk sizes, we only need to test from a larger parallelism degree, since small chunks are more parallelism friendly and it is unlikely to achieve higher performance at a low parallelism degree as large chunks. Figure 15 gives such an example: 4 parallel jobs for 4MB, 8 parallel jobs for 1MB, and 16 parallel jobs for 64KB are the best choices.

To illustrate the actual effect of parallelization to prefetching, we implement an adaptive prefetching algorithm in our emulator. We adopt the history-based prefetching window to determine the prefetching granularity, which is similar to the file prefetching scheme used in Linux kernel. A prefetching window is maintained to estimate the best prefetching degree. The initial window size is 0 and is enlarged based on the detected sequentiality of observed accesses. Assuming chunk `n` of an object is requested, if chunk `n-i`, chunk `n-i+1`, $\cdots$, chunk `n-1` ($1 \leq$`i`$\leq$`n`) are detected to be sequentially accessed, the size of the prefetching window grows to 2^{i-1}. We set the maximum prefetching window size (i.e., parallelism degree of prefetching) for all chunk sizes (i.e., 64KB, 1MB, 4MB) to 8.

The performance comparison of no-prefetching and prefetching are shown in Figure 16 and Figure 17. We can see that, with prefetching, the optimal chunk size is 1MB. Obviously, small chunk size benefits more from the prefetching (as we see in the prior sections, small objects benefit more from parallelism), and the relative benefits decrease as the chunk size increases (see Table VI).

Chunk Size	Read Lat. Red.	Write Lat. Red.
64KB	70.4%	31.1%
1MB	56.9%	24.6%
4MB	22.6%	-1.2%

TABLE VI
AVERAGE LATENCIES REDUCTION CAUSED BY PREFETCHING

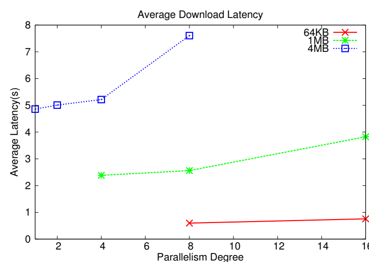


Fig. 15. Avg. Download Latency

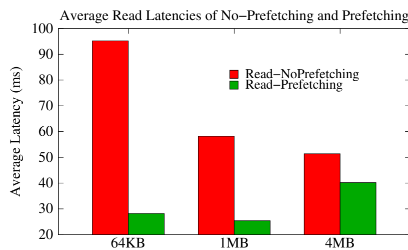


Fig. 16. Avg. Read Latency Comparison

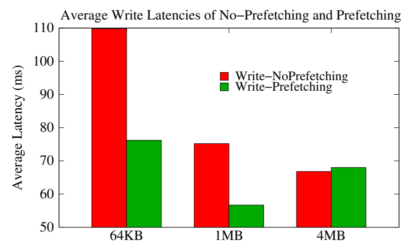


Fig. 17. Avg. Write Latency Comparison

Surprisingly, with prefetching, the average write latency of 4MB increases by 1.2%. This means that the prefetching granularity in our experiment is so aggressive that the negative effects of prefetching overweight the benefits. The negative effects may result from two factors. First, a lot of unnecessary data is prefetched so that the cache efficiency is reduced, leading to a lower cache hit ratio. Second, the competition of parallel prefetching threads may increase the average downloading latency (i.e., the average penalty of cache miss). Specifically for the case of the average write latency of 4MB, the performance degradation is mainly caused by the second factor since the cache hit ratio remains high (close to 98.7%). As a rule of thumb, we should set a small prefetching degree for large chunk sizes (e.g., 4MB) to avoid the intensive competitions of the parallelized downloading threads. For example, we can limit the growing speed of the prefetching window, or cap the maximum prefetching window size. On the contrary, the prefetching granularity of small chunk sizes (e.g., 64KB) can be more aggressive. This also confirms our speculation about the proper parallelism degrees for different chunk sizes (i.e., 16 for 64KB chunks, 4 for 4MB chunks).

In summary, our case studies on prefetching and caching further show that the real-world implementation of client-side management should carefully consider the factors that we have studied in the prior sections, particularly parallelism degree and request size. Other issues, such as client capabilities and geo-distances would also inevitably further complicate the design consideration.

VI. System Implications

With these experimental observations, we are now in a position to present several important system implications. This section also provides an executive summary of our answers to the questions we asked earlier.

Appropriately combining request size and parallelism degree can maximize the achievable performance. This is sometimes a tradeoff between the two factors. By combining the chunking/bundling methods with parallelizing I/Os, the client can enhance bandwidth in two different ways: we can increase the parallelism degree for small requests or increase the request size at low parallelism degree. Both can achieve comparable bandwidth, but interestingly, we also find that compared to increasing parallelism degree, increasing the request size can achieve another side benefit: reduced CPU utilization. This means that for some weak-CPU platforms, such as mobile systems, it is more favorable to create large requests with a low parallelism degree. On the other hand, we should also consider several related side effects of bundling/batching small requests. For instance, if part of a bundled/batched request failed during the transmission, the whole request would have to be re-transmitted. Also, it is difficult to pack a bunch of small requests to different buckets or data centers together. In contrast, parallelizing small requests is easier and more flexible. Therefore, there is no clear winner between the two possible optimization methods (i.e., creating large requests and parallelizing small requests). An optimal way may vary from client to client and from service to service, but we can still use some general principles to guide us in making a decision. For example, as we find that small requests demand a high parallelism degree, if we know the proper parallelism degree on a certain client for 1MB is 8, and 4 for 4MB (we can obtain these combinations via simple measurement or experience), it is reasonable to infer that the proper parallelism degree for 2MB should be between 4 and 8. To avoid the worst situation, a rule of thumb is to make a conservative choice, if uncertain.

The client's capability has a strong impact to the perceived cloud storage I/O performance. CPU, memory, and storage are the three most critical components determining a client's capability. Among the three, CPU plays the most important role in parallelizing small requests, while memory and storage are critical to large requests, especially large download requests. A direct implication is that for optimizing the cloud storage performance, we must also distinguish the capabilities of clients, and one policy will not be effective in all clients. Due to the cross-platform advantage, many personal cloud storage applications can run on multiple platforms (from PCs to Smartphones). Such distinction among clients will inevitably affect our optimization policies. For example, for a mobile client with a weak CPU, we should avoid segmenting objects into excessively small chunks, since it is unable to handle a large number of parallel I/Os, although this is not a constraint for a PC client. Given the diversity of cloud storage clients, we believe that a single optimization policy is unlikely to succeed across all clients.

Geographical distance between the client and the cloud plays an important role in cloud storage I/Os. For cloud storage, the geographical distance determines the RTT. We find that a long RTT has distinct effects to bandwidth and latency. In particular, with a long RTT, we still can achieve a similar peak bandwidth as the case of a short RTT, but

the cloud I/O latency is significantly higher. The implications are two-fold. First, to tackle the long latency issues, it is a must-have to use effective caching and prefetching for latency-sensitive applications. Second, for the clients far from the cloud, we should proactively adopt large request sizes and high parallelism degrees to fully saturate the pipeline and exploit available bandwidth as much as we can. In other words, by sufficiently exploiting the I/O characteristics of cloud storage, if bandwidth is the main requirement (e.g., video streaming), choosing a relatively distant data center of the cloud storage is a viable option and a high bandwidth is still achievable with appropriate client-side optimizations.

In essence, cloud storage represents a drastically different storage model for its diverse clients, network-based I/O connection, and massively parallelized storage structure. Our observations and analysis strongly indicate that fully exploiting the potential of cloud storage demands careful consideration of various factors.

VII. Related Work

Most prior studies focus on addressing various issues of cloud storage, including performance, reliability, and security (e.g., [14], [20], [21], [22], [24], [31], [32], [33], [35], [41], [47], [48]). Some other work studies the design of cloud-based file systems to better integrate cloud storage into current storage systems (e.g., [18], [26], [45]). Our work is orthogonal to these studies and focuses on understanding the behaviors of cloud storage from the client's perspective.

Our work is related to several prior measurement works on cloud storage. Li et al. compared the performance of four major cloud providers: Amazon AWS, Google AppEngine and Rackspace CloudServers [40]. Ou et al. compared a file system client of cloud storage based on CloudFuse with two other IP-based storage, NFS and iSCSI [44]. Bermudez et al. presented a characterization of Amazon's Web Services (AWS) [17]. Copper et al. benchmarked cloud storage systems with YCSB [23]. Meng et al. presented a benchmarking work on cloud-based data management systems to evaluate the effects of different implementation on cloud storage [43]. This work treats cloud storage as a blackbox and focuses on characterizing its I/O behaviors from client's perspective. Several other measurement works focus on the client applications (e.g., [27], [28], [29], [34], [42], [46]). Unlike our study, these prior studies focus on measuring the performance of commercial personal cloud storage clients, such as Dropbox, Wuala, Google Drive, etc. In contrast, we treat the cloud storage as a blackbox and focus on revealing the key factors affecting the interactions of the client and the cloud from the perspective of the client side rather than benchmarking specific cloud storage clients. In fact, we purposely avoid using any specific client tools so that we can minimize the potential interference. Besides object-based storage, some service providers also provide block- and file-level storage services, such as Amazon Elastic Block Store (EBS) [1] and Elastic File System (EFS) [2]. These services are more similar to conventional IP-based storage, such as iSCSI and NFS. In this work, we focus on the HTTP-based object storage, represented by Amazon S3, and we have observed several interesting and unique I/O behaviors.

VIII. Conclusions

We present a comprehensive measurement and quantitative analysis on cloud storage to investigate the critical factors affecting the perceived performance of cloud storage from a client-side perspective. Our experiments show several important characteristics of cloud storage, such as benefits of parallelizing cloud storage I/Os, the latency impact of over-parallelization, the effect of client's capability to achievable performance, and more. Based on these findings, we also present a case study on chunking and parallelization, the two key techniques used for optimizing cloud storage performance on the client. We hope our observations and the associated system implications can provide guidance to help practitioners and application designers exploit various optimization opportunities for cloud storage clients.

Acknowledgment

The authors thank anonymous reviewers for their constructive comments to improve this paper. This work was supported in part by Louisiana Board of Regents under grants LEQSF(2014-17)-RD-A-01 and LEQSF-EPS(2015)-PFUND-391, National Science Foundation under grant CCF-1453705, and generous support from Intel Corporation.

References

[1] Amazon EBS. https://aws.amazon.com/ebs/.
[2] Amazon EFS. https://aws.amazon.com/efs/.
[3] Amazon S3. https://aws.amazon.com/s3/.
[4] Amazon S3 TCP Window Scaling. http://docs.aws.amazon.com/AmazonS3/latest/dev/TCPWindowScaling.html.
[5] Cloud Storage Market by Solutions. http://www.marketsandmarkets.com/Market-Reports/cloud-storage-market-902.html.
[6] Dropbox. https://www.dropbox.com/.
[7] Dropbox Uses Amazon S3 Services for Storage! https://storageservers.wordpress.com/2013/10/25/dropbox-uses-amazon-s3-services-for-storage/.
[8] Google Drive. https://www.google.com/drive/.
[9] OneDrive. https://onedrive.live.com/.
[10] OpenStack Swift. http://www.openstack.org/.
[11] S3 API Reference. https://boto.readthedocs.org/en/latest/ref/s3.html.
[12] S3Backer. https://code.google.com/p/s3backer/.
[13] S3FS. https://code.google.com/p/s3fs/.
[14] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A Case for Cloud Storage Diversity. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC 2010)*, Indianapolis, IN, June 10-11 2010.
[15] Amazon. Amazon S3 Object Size Limit Now 5 TB. https://aws.amazon.com/blogs/aws/amazon-s3-object-size-limit/.
[16] A. Bergen, Y. Coady, and R. McGeer. Client Bandwidth: The Forgotten Metric of Online Storage Providers. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim 2011)*, Victoria, BC, Canada, August 23-26 2011.
[17] I. Bermudez, S. Traverso, M. Mellia, and M. Munafo. Exploring the Cloud from Passive Measurement: the Amazon AWS Case. In *Proceedings of The 32nd IEEE International Conference on Computer Communications (INFOCOM 2013)*, Turin, Italy, April 14-19 2013.
[18] Bessani, Alysson, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, , and P. Verissimo. SCFS: A Shared Cloud-backed File System. In *Proceedings of the 2014 USENIX Annual Technical Conference (ATC 2014)*, Philadelphia, PA, June 19-20 2014.

[19] E. Bocchi, I. Drago, and M. Mellia. Personal Cloud Storage Benchmarks and Comparison. *IEEE Transactions on Cloud Computing*, 99:1–14, 2015.

[20] N. Bonvin, T. G. Papaioannou, and K. Aberer. A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC 2010)*, Indianapolis, Indiana, June 10-11 2010.

[21] C. Brad, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, and Y. X. et al. Windows Azure Storage: a Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011)*, pages 119–132, New York, NY, October 23-26 2011.

[22] F. Chen, M. P. Mesnier, and S. Hahn. Client-aware Cloud Storage. In *Proceedings of the 30th International Conference on Massive Storage Systems and Technology (MSST'14)*, Santa Clara, CA, June 2-6 2014.

[23] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and V. colleagues. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (ACM SoCC 2010)*, Indianapolis, IN, June 10–11 2010. ACM Press.

[24] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom 2015)*, pages 582–603, Sept 7-11 2015.

[25] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch. In *Proceedings of the 2007 USENIX Annual Technical Conference*, Santa Clara, CA, June 2007. The USENIX Association.

[26] Y. Dong, J. Peng, D. Wang, H. Zhu, F. Wang, S. C. Chan, and M. P. Mesnier. RFS - A Network File System for Mobile Devices and the Cloud. In *SIGOPS Operating System Review*, volume 45, pages 101–111, February 2011.

[27] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking Personal Cloud Storage. In *Proceedings of the 2013 ACM conference on Internet measurement conference (IMC 2013)*, Barcelona, Spain, October 23-25 2013.

[28] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Modeling the Dropbox Client Behavior. In *Proceedings of the 2014 IEEE International Conference on Communicationsconference (ICC 2014)*, Sydney, NSW, June 10-14 2014.

[29] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 2012 ACM conference on Internet measurement conference (IMC 2012)*, New York, NY, November 14-16 2012.

[30] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS Tracing of Email and Research Workloads. In *Proceedings of the 2th USENIX Conference on File and Storage Technologies (FAST 03)*, San Francisco, CA, March 2003. The USENIX Association.

[31] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in Globally Distributed Storage Systems. In *Proceedings of 9th USENIX Symposium on Operating Systems Design and Implementation*, Vancouver, Canada, Oct 4-6 2010.

[32] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal. Pesto: Online Storage Performance Management in Virtualized Datacenters. In *Proceedings of the 2nd ACM symposium on Cloud computing (SoCC 2011)*, Cascais, Portugal, October 27.28 2011.

[33] B. D. Higgins, J. Flinn, T. Giuli, B. Noble, C. Peplin, and D. Watson. Informed Mobile Prefetching. In *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys 2012)*, pages 155–158, June 25-29 2012.

[34] W. Hu, T. Yang, and J. N. Matthews. The Good, the Bad and the Ugly of Consumer Cloud Storage. In *ACM SIGOPS Operating Systems Review*, volume 44:3, July 2010.

[35] Y. Hu, H. C. H. Chen, P. P. Lee, and Y. Tang. NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST 2012)*, San Jose, CA, February 14-17 2012.

[36] L. Huan. A Trace Driven Study of Packet Level Parallelism. *IEEE Communications (ICC 2002)*, 4(1):2191–2195, April 28-May 2 2002.

[37] IHS. Subscriptions to Cloud Storage Services to Reach Half-Billion Level This Year. https://technology.ihs.com/410084/subscriptions-to-cloud-storage-services-to-reach-half-billion-level-this-year.

[38] V. Jacobson, R. Braden, D. Borman, M. Satyanarayanan, J. Kistler, L. Mummert, and M. Ebling. RFC 1323: TCP Extensions for High Performance, 1992.

[39] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang. DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Localities. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST 2005)*, San Francisco, CA, December 12-16 2005. The USENIX Association.

[40] A. Li, X. Yang, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (ACM SIGMOD 2010)*, Melbourne, Australia, November 1–3 2010. ACM Press.

[41] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai. Efficient Batched Synchronization in Dropbox-like Cloud Storage Services. In *Middleware 2013*, pages 307–327. Springer, 2013.

[42] T. Mager, E. Biersack, and P. Michiardi. A Measurement Study of the Wuala On-line Storage Service. In *Proceedings of the IEEE 12th International Conference on Peer-to-Peer Computing (P2P 2012)*, Sophia Antipolis, France, Sept 3-5 2012.

[43] X. Meng, Y. Chen, J. Xu, and J. Lu. Benchmarking Cloud-based Data Management Systems. In *Proceedings of the 2nd international workshop on Cloud data management (ACM CloudDB 2010)*, Toronto, ON, October 26–30 2010. ACM Press.

[44] Z. Ou, Z.-H. Hwang, A. Ylä-Jääski, F. Chen, and R. Wang. Is Cloud Storage Ready? A Comprehensive Study of IP-based Storage Systems. In *Proceedings of the 8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'15)*, Limassol, Cyprus, December 7–10 2015.

[45] M. Vrable, S. Savage, and G. M. Voelker. BlueSky: A Cloud-Backed File System for the Enterprise. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST 2012)*, San Jose, CA, February 14-17 2012.

[46] H. Wang, R. Shea, F. Wang, and J. Liu. On the Impact of Virtualization on Dropbox-like Cloud File Storage/Synchronization Services. In *Proceedings of International Workshop on Quality of Service (IWQoS 2012)*, Coimbra, Portugal, June 4-5 2012.

[47] R. Zhang, R. Routray, D. Eyers, D. Chambliss, P. Sarkar, D. Willcocks, and P. Pietzuch. IO Tetris: Deep Storage Consolidation for the Cloud via Fine-grained Workload Analysis. In *Proceedings of the 4th International IEEE Conference on Cloud Computing (IEEE CLOUD 2011)*, Washington D.C., July 2011.

[48] Y. Zhang, C. Dragga, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. ViewBox: Integrating Local File Systems with Cloud Storage Services. In *Proceedings of the 12th USENIX conference on File and Storage Technologies (FAST 2014)*, pages 119–132, San Jose, CA, February 14-17 2014.