Redefining Data Locality for Cross-Data Center Storage

Kwangsung Oh, Ajaykrishna Raghavan, Abhishek Chandra, and Jon Weissman
Department of Computer Science and Engineering
University of Minnesota Twin Cities
Minneapolis, MN 55455
{ohkwang, raghavan, chandra, jon}@cs.umn.edu

ABSTRACT

Many Cloud applications exploit the diversity of storage options in a data center to achieve desired cost, performance, and durability tradeoffs. It is common to see applications using a combination of memory, local disk, and archival storage tiers within a single data center to meet their needs. For example, hot data can be kept in memory using ElastiCache, and colder data in cheaper, slower storage such as S3, using Amazon as an example. For user-facing applications, a recent trend is to exploit multiple data centers for data placement to enable better latency of access from users to their data. The conventional wisdom is that co-location of computation and storage within the same data center is a key to application performance, so that applications running within a data center are often still limited to access local data. In this paper, using experiments on Amazon, Microsoft, and Google clouds, we show that this assumption is false, and that accessing data in nearby data centers may be faster than local access at different or even same points in the storage hierarchy. This can lead to not only better performance, but also reduced cost, simpler consistency policies and reconsidering data locality in multiple DCs environment. This argues for an expansion of cloud storage tiers to consider non-local storage options, and has interesting implications for the design of a distributed storage system.

Keywords

Data Locality; Mutli-DCs; Multi-tiered storage; Wide Area Storage; In Memory Storage

1. INTRODUCTION

Cloud providers offer multiple cloud storage services with different characteristics, such as durability, performance, and cost from data centers (DCs) that are distributed geographically. For example, Amazon's Simple Storage System (S3) provides high durability and low cost but low performance while ElastiCache provides high performance but high cost and low durability through VM instance. Google, Microsoft, and many other cloud providers also provide similar cloud storage services. These storage services enable cloud service providers to serve data to their users without the need to invest in building their own DCs. Users of cloud services are typically dispersed across the globe. To reduce user-perceived latency, several wide-area storage systems [1, 8, 30] replicate data to multiple DCs to get data closer to users. Applications running inside a data center are often still limited to access local data, since the conventional wisdom is that locality of access *within* the local data center is important. In this paper, however, we show that this assumption is not necessarily true if data is stored in other storage tiers[1] besides memory, and when there are other DCs situated within close proximity.

Table 1: Regions of Data Centers

Region (Area)	Cloud Provider
US East (Virginia)	Amazon, Microsoft
US West 1 (California)	Amazon, Microsoft
US West 2 (Oregon)	Amazon, Google
Europe West (Ireland)	Amazon, Microsoft, Google
Asia Southeast (Singapore)	Amazon, Microsoft, Google
Asia East (Tokyo)	Amazon, Microsoft, Google
South America (Sao Paulo)	Amazon, Microsoft

Previous research [3, 19] has argued that data being accessed by an application in a DC can be located on any node within that DC without concern for data locality, due to the improvement in network speeds. The focus of these works is on a single DC where nodes are connected via Local Area Network. However, the density of data centers has been increasing in the recent past; according to datacentermap.com [11], for example, there are 197 DCs in California as of Jan 2015. In this paper, therefore, we ask the follow-on question: *"Can data be located outside the data center without concern for locality?"* If network performance between two DCs is not a bottleneck, data can be stored on another DC without locality concern for better performance, reduced cost and reliability, thus eliminating cross-data center boundaries.

Using experiments on Amazon, Microsoft, and Google clouds, we find that accessing data from storage tiers in a nearby DC can prove to be faster than accessing data from local storage tiers, and in some cases, substantially faster.

[1] In this paper, we use the term storage tier and storage service interchangeably.

BigSystem'15, June 16, 2015, Portland, Oregon, USA.
Copyright © 2015 ACM 978-1-4503-3568-3/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2756594.2756596.

Table 2: Latency (ms) between DCs

Region	US West		US East		Europe West			Asia Southeast	
	AWS	Azure	AWS	Azure	AWS	Azure	GC	AWS	Azure
AWS	-	3.84	-	1.97	-	17.58	16.33	-	1.84
Azure	3.62	-	1.99	-	18.67	-	16.02	1.98	-
GC	-	-	-	-	16.35	16.12	-	-	-

Table 3: Bandwidth (MB/s) between DCs

Region	US West		US East		Europe West			Asia Southeast	
	AWS	Azure	AWS	Azure	AWS	Azure	GC	AWS	Azure
AWS	-	48.75	-	48.13	-	48.38	48.63	-	48.88
Azure	21.62	-	23.63	-	45.25	-	53.5	24.38	-
GC	-	-	-	-	32.38	40.25	-	-	-

That is, if local I/O performance is slow with respect to a given tier, then better performance may be found by going to one or more tiers in nearby DCs. For example, our result shows that if 100KB of data is on Microsoft Azure disk and Amazon AWS memory, retrieving data from AWS memory is 4.37 times faster for an application running on Azure. In effect, this observation allows to expand the radius of locality, opening up new opportunities. This fact can be exploited by storage system developers to design a rich set of software-defined policies to achieve better performance, implement simpler consistency mechanisms and reduce the total cost of storage.

The remainder of this paper is organized as follows. Section 2 illustrate the performance advantage of non-local DC data access through real experiments carried out on Amazon AWS, Microsoft Azure and Google Cloud at several geographic regions. In Section 3, we use insights gained from our experimental results to present possible opportunities and use cases that can leverage our observations. Section 4 shows how opportunities can be realized using real benchmark deployment. Section 5 discusses the challenges in realizing these opportunities. Section 6 reviews related work, while Section 7 concludes our work.

2. EXPERIMENTAL EVIDENCE OF NON-LOCAL DC LOCALITY

To answer the question raised in the previous section, we conducted a series of experiments over the course of several months from Spring to Winter 2014 and we observed consistent results through the year. We used data centers hosted by three cloud vendors, Amazon (AWS), Microsoft (Azure), and Google Cloud (GC) for our experiments. To avoid any confusion, we use the term *'nearby'* to refer to a DC located in the same geographic region (e.g., both DCs in US West-California), and *'remote'* to refer to a distant DC in another region (e.g., US East-Virginia vs. US Central-Iowa).

2.1 Regions for experiments

Cloud providers' operation policy allows us to know whether DCs are nearby or remote as Table 1 shows, but not precise geographic location. We estimated network performance between DCs within a region and found the four regions, US West (California), US East (Virginia), Europe West (Ireland), and Asia Southeast (Singapore), where there are at least two DCs and the network performance among them is relatively better than other regions. Since Google does not allow us to provision a VM on US-West, US-East, and Asia-Southeast regions[2], we will mainly show results from AWS and Azure in this section. Table 2 and 3 show the latency and bandwidth for each region. From Table 2, we can see that latency between AWS and Azure is much smaller than well-known average disk seek time 10 ms [16], US East (< 2 ms), US West (< 4 ms), and Asia Southeast (< 2 ms). For Europe West region, latency between DCs (> 15 ms) is bigger than 10 ms so that network may be a bottleneck for latency sensitive applications. Table 3 shows that network bandwidth is not a bottleneck if application uses data size smaller than 20 MB. To estimate latency we used 'tcpping' [27] rather than 'ping' since Azure does not allow users to use ICMP protocol. We used 'Iperf' [14] to measure network bandwidth.

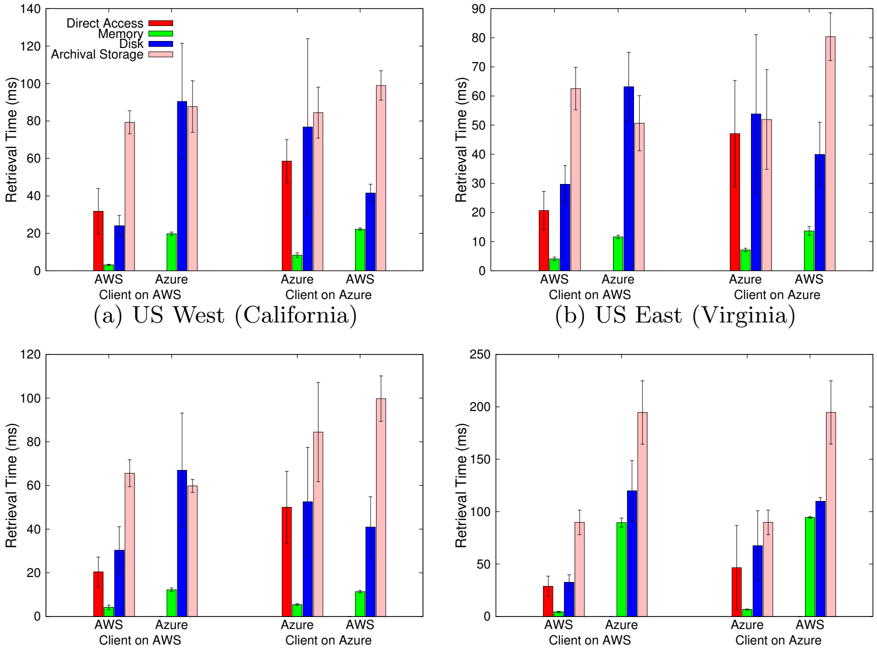
2.2 Experimental Setup

We deployed servers that serve data from both memory and local disk in AWS, Azure, and GC at several regions, US West (California), US East (Virginia), Europe West (Ireland), and Asia Southeast (Singapore), as we stated in 2.1. We also deployed clients that estimate time to retrieve various size of data from three tiers (disk, memory, and archival storage) based on the following types of access:

- *Process-local*: Retrieving data from node's disk through file system.
- *Machine-local*: Retrieving data from the same node's disk and memory but through another process via socket (as an IPC).
- *DC-local*: Retrieving data from another node's disk and memory within the same DC via socket and from local DC's archival storage.
- *Nearby-DC*: Retrieving data from nearby DC node's disk and memory via socket and nearby DC's archival storage.

Since the performance of *Machine-local* is similar to *Process-local* and *DC-local*, we omit results for it for space reasons. We chose VM instances based on the price, m1.medium ($0.095/hr) for AWS and Basic_A2 ($0.094/hr) for Azure which have 1 ~ 2 core(s) and 3.75 GB memory. The prices are based on the US West (California) region. Note that,

[2] Google only allows users to spawn VM instances on US Central, Asia East, and Europe West as of Jan 2015.



(a) US West (California)

(b) US East (Virginia)

(c) Asia Southeast (Singapore)

(d) Europe West (Ireland)

Figure 1: Elapsed time to retrieve 100KB data from all regions.

we could see similar result patterns regardless of VM instance size except micro instance of which performance can be throttled significantly. This is because these experiments are not influenced much by CPU performance or memory size as a client retrieves files one by one from each server with single process and thread. To limit influence of cache for disk access, we restricted size of available memory during the experiments and cleared instances' buffer cache after reading data from disk. All experimental results are an average of 10 runs, plotted with 95% confidence intervals.

2.3 Local vs. Nearby DC

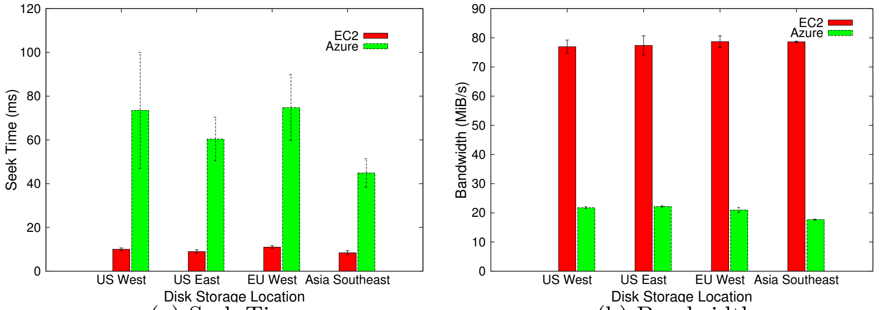
Figure 1 shows the elapsed time for clients on US West, US East, Asia Southeast and Europe West to retrieve 100KB data from different tiers across different servers. Except Europe West region where the latency between DCs is higher than other regions, clients can retrieve data in nearby DC's memory 1.61 ∼ 4.37 times faster than local DC's disk and 3.8 ∼ 7.4 times faster than local DC's archival storage. Local memory in local DC not surprisingly always provides the best performance. Thus, *non-local data access can be faster than local access to slower tiers.*

Interestingly, the results also indicate that *a nearby DC may offer better performance even from the same tier.* That is, clients running on Azure on US West, US East, and Asia Southeast can retrieve data in AWS disk stroage 1.17 ∼ 1.41 times faster than local Azure disk. To verify these results, we estimated the disk performance for each cloud service provider. Figure 2 shows that AWS disk provides better performance, lower seek time (ms) and higher bandwidth (MiB/s), than Azure disk from all regions. We also observed similar disk performance patterns regardless of VM instance size (cost).

For archival storage, we see opposite results. That is, clients running on AWS on US East and Asia Southeast can retrieve data in Azure archival storage 1.1 ∼ 1.23 times faster than local AWS archival storage. In US West region, the client on AWS retrieves data slightly slower but within the margin of error from Azure archival storage than local AWS archival storage. Such performance difference may come from different implementation of disk (archival) storage services as we will discuss later on section 5. Though these results cannot be generalized to all multiple DCs environment, they show the opportunity that we can find better performance even from the same storage tier. Lastly, the results do show the limits of using non-local DC's storage. That is, retrieving data located from a *'nearby'* DC is not always better than a local DC as Figure 1(d) shows. Thus, determining the boundary between *'nearby'* and *'remote'* is an open question.

2.4 Varying Data size

Another important consideration is the data size. Figure 3 shows the performance trend for access times from local (intra-DC) disk vs. nearby (non-local DC) memory for different data sizes when a client is running on AWS US East: 1KB, 10KB, 50KB, 100KB, 1MB, 5MB and 10MB. If the data size is less than 1MB, we see considerable performance improvement, 3.05 times for 1KB, 2.71 times for 10KB, 1.62 for 50KB, 1.64 times for 100KB, 1.18 times for 1MB respectively. When a client is running on Azure, we see much more performance improvement because Azure disk performance is worse than AWS as we shown in Figure 2. However, we omit the result because there is too much performance variance in Azure disk to be generalized. The data size boundary, in this experiment 1MB, can be increased, when we deploy the server on bigger VM instance e.g., from Basic_A2



(a) Seek Time

(b) Bandwidth

Figure 2: Disk seek time and bandwidth of VMs on each region.

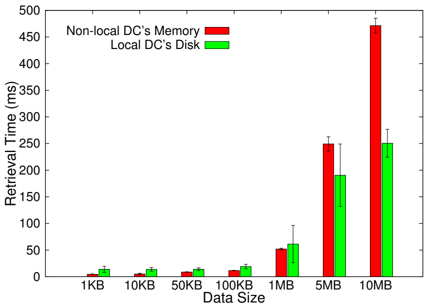


Figure 3: Elapsed time to retrieve data of different sizes from Local (AWS US East) Disk and Nearby (Azure US East) Memory.

($0.094/hr) to Standard_A2 ($0.12/hr) on Azure. This is because cloud providers throttle the network and storage performance based on VM instance size. Fortunately, for many cloud applications, e.g., Web servers, small data size is often the case. Much existing work [4, 15, 17] has shown that many Web servers, Blog servers, and file systems handle data less than 1MB in size. Thus, if median size of data handled by the cloud application is below 1MB, considering nearby DC storage tiers may be highly attractive.

We summarize the key observations from our experiments below:

- Accessing data in main memory, in a nearby DC is faster than accessing data from either disk or archival store in the local DC.
- Accessing data from disk (archival) storage in a nearby DC can be as fast as accessing data from disk (archival) in local DC.
- These trends hold for data sizes up to about 1MB (can be increased), which encompasses many common applications.

These observations suggest the need to revisit storage policies such as those for consistency and replication. For example, the first point suggests that fewer geographically-dispersed replicas serving data requests can be maintained as long as they are kept in a faster tier such as memory - this can reduce the overhead of consistency protocols to keep a large number of replicas synchronized. The second point suggests that backup copies of a data file can be stored in the same tier of non-local DCs for a cheaper price or even for better performance without having to worry about data locality.

3. USE CASES AND OPPORTUNITIES

Having provided experimental evidence that shows non-local data access (from a nearby DC) may be better than local access, we next present possible opportunities and use cases derived from our observations. One obvious question could be: *why not simply store all data in a huge memory tier in a single DC?* The reason is that data replication cannot be avoided for reasons such as fault tolerance, improved durability, and end-user locality. We present a set of use cases which highlight the need for replicating data across multiple DCs and where our approach may provide additional optimization. Note that all use cases assume that DCs in the same region are close to each other unless otherwise stated; previous work [30] suggests that this is a reasonable assumption.

Simpler Consistency Policy

In a distributed storage system, one of the biggest challenges is dealing with data consistency among replicas. With increased number of distributed replicas, weaker consistency models like eventual consistency need to be employed for scalability. This is because of the high overhead of the number of messages that need to be exchanged to impose stronger consistency models between replicas to make all replicas consistent. Based on our observation of the wider locality envelope, we can relieve this constraint by replicating data to the memory tiers of fewer DCs serving more clients across fewer replicas. If we were to replicate across K DCs and require strong consistency, an update in one DC would need to be propagated to all K DCs. Instead, given the locality envelope, data can perhaps be replicated to L DCs ($L < K$) with minimal impact on the latency experienced by clients in this region. Lowering the number of replicas reduces the network traffic needed for data consistency.

Hot and Cold Data

Data can be replicated to multiple DCs for several reasons such as reduced latency, increased durability, and increased fault tolerance. Suppose there are two replicas in the same region at DC A and B. Now suppose that data in a given DC is placed in memory if it is hot (recently accessed) or a

Table 4: Time to spawn a VM on US East region measured from when the request was initiated to successful login to the VM and memory size

	Micro	Small	Medium	Large
AWS	37 secs - 615MB	57 secs - 1.7GB	38 secs - 3.75GB	40 secs - 7.5GB
Azure	121 secs - 750MB	108 secs - 1.75GB	125 secs - 3.5GB	98 secs - 7GB
GC	33 secs - 600MB	49 secs - 1.7GB	49 secs - 3.75GB	34 secs - 7.5GB

slow tier (if it is cold) to reduce cost, as is fairly common. At a given point in time, a particular data object O may be hot in DC A and cold in DC B based on client access. If an application running in DC B tries to access O in DC B then it will pay the price of access to a slow tier. Instead, the system can more efficiently retrieve O from DC A since it is located in the faster memory tier. To achieve this benefit, metadata indicating the location of the data in the remote DC needs to be kept up to date and the system must ensure that the replica at DC A does not get overloaded.

Higher Availability

Since downtime implies revenue loss, applications need to be able to survive failures. Replicating within a DC still leaves data vulnerable to DC-level outages as noted in [2, 5]. Thus, replicating data across independent fault domains and DCs may be desired to achieve high availability. For instance when an application cannot access data from DC A because of an outage, the application can continue normal operation by retrieving data from DC B. By using a faster storage tier at the nearby DC, it is possible to minimize the performance penalty.

Expanding the Memory Tier

In the cloud, it is difficult to estimate beforehand the appropriate number of VMs required to support application workload. For example, if an application running in the cloud needs to expand the capacity of the memory tier (like Memcached), it will have to spawn VMs on-the-fly. However, as can be seen from Table 4, this may be expensive, particularly for Azure. Moreover spawning a new VM for memory also can be failed not because of DC's failure but cloud providers' capacity for each VM instance. For example, we could see that Amazon EC2 rejects to create a new VM instance with an error message "Error: InsufficientInstanceCapacity" [13]. In either case, the application can more quickly acquire memory storage resources in a nearby DC, that has a faster VM spawn time and available instance capacity.

Competitive Pricing

Cloud providers offer different storage services and VM instance types which have different storage capacities at different prices. Thus, if a user is near to multiple DCs, one has more options to choose different storage service or VM instance from different cloud provider. For example, the user can choose 13 different VM instance (4 from AWS, 5 from Azure, and 4 from GC) with various storage capacities (memory and disk) at the price less than $0.1 per hour. Besides, some VM types allow users to choose small size of SSD instead of huge size of disk as a main storage and thus users have even more options based on their requirements. For different workloads, cloud providers also offer optimized instances for them. For example, Amazon, Microsoft, and Google offer VM instances for memory-intensive workloads, Optimized compute_D13 (8 cores, 56GB memory, $617 per month) from Azure, r3.2xlarge (8 cores, 61GB memory, $521 per month) from AWS, and n1-highmem-8 (8 cores, 52GB, $309 ~ $440 per month based on average usage) respectively. Thus, one can choose to use a VM in a nearby DC if it is cheaper than a local VM and the network is not a bottleneck. For example, if an application on Microsoft's DC A needs about 50 GB memory, it can use an r3.2xlarge on Amazon's DC B or n1-highmem8 on Google's DC C rather than an A7 VM on DC A for a cost reduction of $96 per month (a saving of 15.5%) or $177 ~ $307 per month (a saving of 28.7% ~ 50% based on usage) while retaining similar performance. Of course, the amount of network traffics between DCs should be considered to get maximized cost-benefit as cloud providers charge for it.

4. WEB APPLICATION CASE STUDY

In this section, we will show how some of the opportunities outlined in the previous section can be realized for a Web application, using a popular open-source benchmark RUBiS [25]. RUBiS is a multi-component web application that implements functions of an auction site EBay.com, selling, buying, bidding, commenting and so on. We use Apache and PHP for front-end web server and MySQL for back-end database. It may not fully capture the complexities of today's web application as previous work [7] pointed out, but it is sufficient to show benefits from using non-local DC's resource.

For this experiment, we deployed Apache and MySQL on each cloud provider's DC at each region. Then, we configured RUBiS to use MySQL running on 1) Local DC with disk, 2) Local DC with memory, and 3) Nearby DC with memory. For the first deployment, MySQL uses local disk like standard MySQL deployment. For the second and third deployments, we created ramdisk and configured MySQL to use it. Like our previous experiments, we restricted available memory of VM instance where MySQL is running on to limit influence of buffer cache for disk access. All static contents requested through web server are stored on memory to avoid any disk access for web contents. The primary metric measured by the benchmark is throughput (requests/second). The database we used was populated with 1,000,000 items and 1,000,000 users and its size is around 2GB. We ran client-browser emulator which simulates 300 users on separate instance on the DC where Apache is running. We chose the cheapest VM instances which have 2 CPU cores and at least 3.5GB memory from each cloud providers for both RUBiS and browser emulator, t2.medium ($0.056/hr) for AWS, Standard A2 ($0.12/hr)[3] for Azure, and n1-standard-2 ($0.106/hr) for GC. The prices are based on the Europe West region. We used MySQL v.5.5.40 (Ubuntu), Apache2 v2.4.7 (Ubuntu) and PHP module v.5.5.9-1. The benchmark was run 300 seconds, with 120 seconds for ramp-up, 60 seconds for ramp-down.

From our previous experiment, we did not see performance benefit from Europe West region because of higher network

[3] We chose this instance instead of cheaper one Basic A2 ($0.102/hr) because of CPU performance.

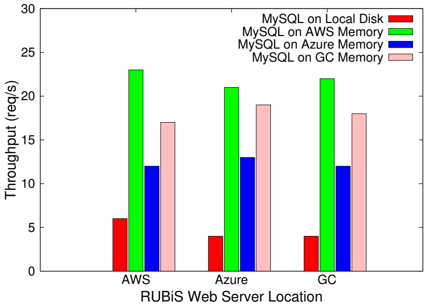


Figure 4: Throughput from various MySQL database storage locations in Europe West region.

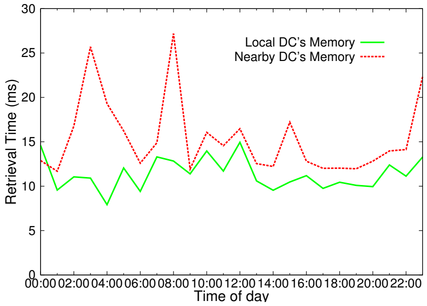


Figure 5: Performance variance over time.

latency than other regions. Yet, Figure 4 shows that MySQL running on non-local DC with memory can provide much better throughput than MySQL running on local DC with local disk in Europe West region. The result shows that offloading I/O access to nearby DC's memory even with somewhat expensive network cost can provide much better performance when an application requires fully random data access to a local slow tier. Of course, we also observed the same pattern of results from all other regions where the network performance is better than this region.

Surprisingly, the result shows nearby DC's memory can be better than local's memory. That is, MySQL on AWS with memory results in the best throughput regardless of web server location. This is because throughput is affected by not only data location but also CPU performance in this case. From the CPU performance benchmark, we see that AWS (t2.medium) provides the best performance. GC (n1-standard-2) provides similar but slightly worse than AWS but better than Azure (Standard A2). Unlike a single DC environment where RDMA (Remote Direct Memory Access) can be used for improving performance to access non-local memory, there is no such option in multiple DCs environment. Thus, CPU performance also needs to be considered to get the most benefits of performance from non-local memory. One interesting and important finding in this experiment is that we can use VM instances in nearby DCs which have better CPU performance with similar memory size even with much cheaper price. In this experiment, for example, applications running on Azure and GC in Europe West can consider to use AWS' resource for both better performance and reduced cost (53% and 47% cheaper than Azure and GC respectively).

5. CHALLENGES

Having shown the potential of increasing the locality envelope, we now present some research challenges to realizing a practical deployment that can take advantage of the above opportunities. We note that the benefits described above do not come for free. Using multiple DC resources introduces many complexities in terms of performance variation, data consistency, data durability, and cost concerns, including wide-area network characteristics as well as network performance variation. We address some of these issues here.

Infrastructure Dynamics

As numerous papers [21, 26] mention, cloud storage services do not provide consistent performance over time. Benson et al [6] found that this is also true for network bandwidth between DCs. We also have confirmed this from experiments. Figure 5 shows the elapsed time for a client on Amazon in N. Virginia to retrieve 100KB data from local and nearby DC memory storage through the course of a single day. Interestingly, the client can experience performance variations even within local DC memory, though the network variation has larger peaks. We also observe that network performance variance can result in high worst case fetch times at different times of the day (e.g., between 00:00 AM and 10:00 AM in Figure 5). Thus, a storage system which utilizes nearby DC resources should be made aware of the potential for such variations in storage and network performance. A possible solution is to build a model, using historical data, that can predict performance at a given time of day. Based on the model, the storage system can avoid the use of nearby memory at certain times (in this case, between 00:00 AM and 10:00 AM).

Application Dynamics

As we have shown, the performance benefit depends on the size of data. The size of accessed data can change dynamically and storage systems need to be able to adapt to these changes in access sizes [20]. In our case, if application data size becomes larger than 1MB (can be increased based on VM instance), the experiments show that the latency to access data from a faster tier in a nearby DC tends to converge to the latency to access data from a local slower storage tier (Figure 3). Moreover, if the data size is too big, memory is not a good choice because it is much more expensive than any other storage tiers. For dynamically changing workloads, the following questions arise: "What is the anticipated workload and how can we detect whether it is temporary or persistent?", and "When should data be replicated vs accessed remotely?".

Simple Storage Abstraction for Applications

Even within a single DC, each storage service has a different interface and price policy based on its characteristics. As an example, Amazon offers ElastiCache which provides high performance but less durability and high cost, and it offers Simple Storage Service (S3) which provides high durability and low cost but low performance. Thus, the application should handle such complexities to get composite benefits of each storage service. For this reason, two open source web applications that use multiple storage services, WordPress [29] and Moodle [18], have 1000-5000 lines of code only to manage data across different storage services. The situation will be even more complicated if the storage system

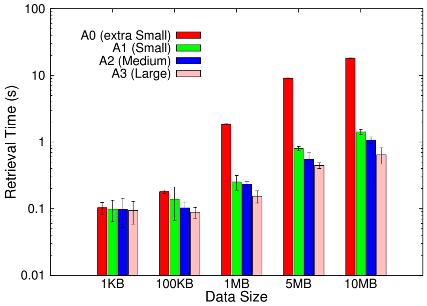


Figure 6: Elapsed time (log scale) for a client on AWS US East to retrieve different size of data on memory from various type of VM on Azure US East.

runs across multiple DCs provided by different cloud vendors. In this case, we can expect even more diverse storage interfaces, pricing policies, and performance variation. It is, therefore, necessary to provide higher level storage abstractions to application developers. The abstractions should hide the above complexities and yet are expressive enough to capture user requirements (e.g., desired read and write latencies, total cost budget, required durability, consistency level, etc.). Our previous work, Tiera [23], provides such storage abstractions for a single DC and extending them to the multiple DCs environment is an interesting area of future work.

Discovering nearby DCs

From the period for our experiments, we saw that the number of DCs have increased very quickly in all regions. For example, the number of DCs on California was 171 on Sep 2014, but now it becomes 197 on Jan 2015. Thus, one can have denser DCs on a region as many DCs have been being built on the region. However, it may be difficult to find nearby DCs within the region if cloud provider's policy does not allow users to know its DCs location precisely for their services (like Google). Moreover, though one discovers that there are many DCs in a specific region within a short physical distance, one may not conclude that those DCs are *nearby* in terms of network performance because network performance is not determined by only physical distance but also several facts such as peering relation among DCs, aspects of the network access-link delays and route policy as previous work [9] pointed out. For example, even if there are two DC A and B on the same region (X) and one DC (C) is on another region (Y), A may find C as a *nearby* DC from region (Y) while A regards B as a *remote* DC because of network peering relation or different routes policy between DCs. The storage systems using multiple DCs should consider many aspects not just distance to discover nearby DCs. One simple solution can be estimating network latency and bandwidth to find nearby DCs as we have shown from Table 2 and 3, though it is expensive in multiple DCs environment (as mentioned, there are 197 DCs only in California).

Cloud Providers' Implementations and Policies

Even if one is able to locate a nearby DC to which a high speed network connection exists, performance benefits might not be realized if the DC provides storage that are themselves slow. That is, same points of storage of cloud providers may provide different performance. For example, our experiments show that AWS' disk provides much better performance than Azure's. This can be because of different implementation of each cloud provider for their own services or because of dynamics of DC as we mentioned at beginning of this section. For neither cases, one cannot get the performance benefit from a nearby DC. Thus, it is required to know the baseline performance of each storage tier and keep monitoring the performance to get benefits. Besides, each cloud provider throttles network and storage bandwidth based on VM instance size as Figure 6 shows. In addition, Table 3 shows network bandwidth throttling ($<$ 22MB/s) in Azure while the network latency is relatively low. Lastly, while we conducted experiments from Jan to Dec 2014, we see that each cloud provider changes (luckily deducts) their price for the services and introduce many new VM instance types. This fact also imposes complexities to storage system developers to choose appropriate size of VM and make them keep monitoring and understanding each cloud providers' policies which are important facts to get maximized performance benefits from using multiple DCs' resources with reduced cost.

6. RELATED WORK

Data Locality

Recent research work has shown that data locality within a DC is irrelevant, given the bandwidth of current DC networks. Ananthanarayanan et al. [3] illustrate how for even large sequential workloads, limited disk bandwidths become a bottleneck. They also show that accessing data from a remote node's memory can provide better performance than reading data from local disk. Nightingale et al. [19] argue that data locality optimizations are unnecessary with the advent of full bisection bandwidth networks, and that these optimizations can prevent efficient use of resources. We show that data locality may also be irrelevant in multiple DCs environment, and accessing data over the network from same or faster storage resource in a nearby DC can be faster than using a slower local storage tier.

In Memory Storage

Many previous works utilize memory to improve performance. Cooperative Caching [10] tries to use idle remote node's memory to improve file system performance. Many recent storage systems, like Redis [24] and RAMCloud [22] aggregate memory resources from many nodes and present it as a common storage pool to applications. But these approaches utilize memory within a single DC, unlike our proposal in this paper. All the above systems assume that network is not a bottleneck. We have shown that this assumption also can be true for multiple DCs environment.

Wide Area Storage

Many storage systems utilize multiple DCs. Volley [1] performs automated data placement across distributed DCs using diurnal and weekly users' data access patterns to reduce user perceived-latency and to minimize costs associated with inter-DC traffic. Google introduced Spanner [8] which focuses on data distribution across distributed DCs while maintaining externally-consistent distributed transactions for their internal applications. Unlike these storage systems using a single cloud provider's DCs, SPANStore [30] tries to utilize multiple providers' DCs rather than a single provider's to get higher DC density to deliver data closer to users with reduced cost, much like a content delivery network. Our work shows that additional benefits to latency, cost, and consistency can be obtained by simple DC selection by considering the performance of the storage tier hierarchy across DCs. It should also be noted that unlike other previous works, we are looking at latencies from the perspective of an application *within* the cloud and not only from users outside a DC.

Multi-Tiered Storage
Many previous works [12, 28] try to utilize multiple tiers (e.g., HDDs vs. SSDs and local vs. cloud storage) for getting composite benefits of multiple storage tiers. In our previous work Tiera [23], we explored building a storage framework that helps applications build a tiered storage system consisting of local DC's memory resources, for better performance, and persistent storage service like S3 or EBS. While Tiera focused on a single DC, in this paper, however, we consider to use storage tiers on multiple cloud providers to get benefits like better fault tolerance, simpler consistency, less locality concerns and reduced cost.

7. CONCLUSION

Using multiple storage services is quite common for applications deployed in the cloud. In addition, by replicating data closer to users, clouds can provide low latency while achieving high data durability and fault tolerance. In this paper, we go further and show that we can exploit differences in the storage hierarchy across DCs to expand the locality envelope, thus eliminating cross-data center boundaries. We have conducted experiments on Amazon AWS, Microsoft Azure, and Google Cloud which show that accessing faster storage tiers in nearby DCs can give better performance than accessing slower tiers within the local DC. We showed the limits of this benefit based on data size and inter-DC latency. We then presented numerous scenarios that can directly benefit from our observations, resulting in a rich set of software-defined policies to achieve better performance, simpler consistency and reduced cost of storage. We also described the challenges, such as infrastructure and application dynamics, that must be overcome to fully realize this potential.

8. ACKNOWLEDGMENT

The authors would like to acknowledge grant NSF CSR-1162405 that supported this research.

9. REFERENCES

[1] S. Agarwal et al. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *NSDI*. USENIX Association, 2010.

[2] Amazon Cloud Service Goes Down and Takes Popular Sites With It. `http://alturl.com/m3a8n/`.

[3] G. Ananthanarayanan et al. Disk-locality in datacenter computing considered irrelevant. HotOS'13, Berkeley, CA, USA, 2011. USENIX Association.

[4] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5), Oct. 1997.

[5] Azure outage. `http://alturl.com/7idm2/`.

[6] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. IMC '10, New York, NY, USA, 2010. ACM.

[7] E. Cecchet et al. Benchlab: An open testbed for realistic benchmarking of web applications. WebApps'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.

[8] J. C. Corbett et al. Spanner: Google's globally-distributed database. OSDI'12, Berkeley, CA, USA, 2012. USENIX Association.

[9] F. Dabek et al. Vivaldi: A decentralized network coordinate system. SIGCOMM '04, pages 15–26, New York, NY, USA, 2004. ACM.

[10] M. D. Dahlin et al. Cooperative caching: Using remote client memory to improve file system performance. 1994.

[11] Data Center Map. `http://www.datacentermap.com/`.

[12] J. Guerra et al. Cost effective storage using extent based dynamic tiering. FAST'11, pages 20–20, Berkeley, CA, USA, 2011. USENIX Association.

[13] Instance Capacity. `http://alturl.com/afmoe`.

[14] Iperf. `https://iperf.fr/`.

[15] M. Jeon et al. Workload characterization and performance implications of large-scale blog servers. *ACM Trans. Web*, 6(4), Nov. 2012.

[16] Known disk seek time. `http://alturl.com/rnik2`.

[17] A. W. Leung et al. Measurement and analysis of large-scale network file system workloads. ATC'08, Berkeley, CA, USA, 2008. USENIX Association.

[18] Moodle. `https://moodle.org/`.

[19] E. B. Nightingale et al. Flat datacenter storage. OSDI'12, Berkeley, CA, USA, 2012. USENIX Association.

[20] R. Nishtala et al. Scaling memcache at facebook. NSDI'13, Berkeley, CA, USA, 2013. USENIX Association.

[21] Z. Ou et al. Exploiting hardware heterogeneity within the same instance type of amazon ec2. HotCloud'12, Berkeley, CA, USA, 2012. USENIX Association.

[22] J. Ousterhout et al. The case for ramclouds: scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review*, 43(4), 2010.

[23] A. Raghavan, A. Chandra, and J. Weissman. Tiera: Towards flexible multi-tiered cloud storage instances. Technical Report TR 14-003, University of Minnesota - Department of Computer Science, Jan 2014.

[24] Redis. `http://redis.io/`.

[25] RUBiS Web site. `http://rubis.ow2.org`.

[26] D. Shue, M. J. Freedman, and A. Shaikh. Performance isolation and fairness for multi-tenant cloud storage. OSDI'12, Berkeley, CA, USA, 2012. USENIX Association.

[27] tcpping. `http://alturl.com/azw8n`.

[28] M. Vrable, S. Savage, and G. M. Voelker. Bluesky: A cloud-backed file system for the enterprise. FAST'12, Berkeley, CA, USA, 2012. USENIX Association.

[29] WordPress. `http://wordpress.org/`.

[30] Z. Wu et al. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. SOSP '13, New York, NY, USA, 2013. ACM.