CONTENT-AWARE RESOURCE SCHEDULING FOR COMMERCIAL AND PERSONAL CLOUDS

BY

SOBIR BAZARBAYEV

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor William H. Sanders

ABSTRACT

Organizations of all sizes are shifting their IT infrastructures to the cloud because of its cost efficiency and convenience. Because of the on-demand nature of Infrastructure as a Service (IaaS) clouds, hundreds of thousands of virtual machines (VMs) may be deployed and terminated in a single large cloud data center each day. At the same time, mobile devices are becoming the primary personal computing platform for hundreds of millions of consumers. Tablets and smart phones are becoming more and more common, and are replacing other forms of computation for most day-to-day tasks.

In this thesis, we present solutions to lower costs of VM deployments in data centers. Also, we present cost-effective personal cloud storage service for mobile devices which uses similar techniques as in VM deployments in data centers. First, we present a content-based scheduling algorithm for the placement of VMs in data centers. We take advantage of the fact that it is possible to find identical disk blocks in different VM disk images with similar operating systems by scheduling VMs with high content similarity on the same hosts. That leads to significant savings in data center network utilization and congestion, by lowering the amount of data transfer associated with deployment of VMs.

Second, we present a personal cloud storage service for mobile devices. Much as there is content similarity between VM disk images that share the same OSs, multiple devices that have the same owner tend to have similar content. We use that content similarity among an individual's multiple mobile devices to minimize the network and storage costs of the personal cloud storage service.

To my family and my late grandmother, for their love and support.

ACKNOWLEDGMENTS

The work described here was performed with funding from AT&T Labs Research.

First and foremost, I thank God for the wisdom and perseverance that he has bestowed upon me during this research project.

I thank my parents Fahriddin and Mukadas, my sister Shakhnoza, and my brother Johongir for their endless love, support, encouragement, and company. I thank my parents for working so hard to make it possible for me and my sister to pursue our degrees at the same time.

I thank my adviser, Professor William H. Sanders, for his technical advice and his continuous support for this project. His belief in the value of this research was an essential part of its success.

I would like to thank the researchers of AT&T Labs Research, Dr. Matti Hiltunen, Dr. Kaustubh Joshi, and Dr. Richard Schlichting, for their support, patience, and collaboration through my graduate research. It has been both a privilege and a pleasure to work with them.

I thank members of the PERFORM research group, Ahmed Fawaz, Ken Keefe, Robin Berthier, Gabe Weaver, Eric Rozier, Michael Ford, Doug Eskins, Sankalp Singh, David Grockocki, Craig Buchanan, Carmen Cheh, and Uttam Thakore, for their friendship, support, and invaluable insights. I thank the members of the group for making our office a pleasant and intellectually simulating environment. Some of the closest friendships I have formed have been within this group. I thank Jenny Applequist for the many hours she has spent providing me with valuable editorial comments on conference papers and this thesis.

I gratefully acknowledge valuable technical advice from Derek Dagit during the early development of this project.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The way we do computing has come to look very different from what it used to be. In the past, personal computers were the primary means of computation for most people. With the evolution of cloud computing, the means of computation now ranges from rental of computation resources from data centers to use of mobile devices to perform most day-to-day tasks. Cloud computing in the form of IaaS has led organizations of all sizes to ditch their privately owned and supported IT infrastructures and move to the cloud. To keep up with the rapid rise in demand, cloud service providers are upgrading their core network infrastructures (e.g., by deploying expensive, specialized 10GbE switches), leading to high network costs. At the same time, we have seen an equally rapid rise in the popularity of mobile devices, like smart phones and tablets. To support the high number of devices, cellular companies offer data plans at very high prices. In this thesis, we present solutions that can be used in both data centers and mobile devices to lower network costs associated with common tasks. Specifically, we address deployment of VMs in data centers, and backup and synchronization to cloud storage for mobile devices.

Large cloud service providers like Amazon Web Services (AWS), Microsoft Azure, and Rackspace have made it very cost-effective for companies to have their services hosted on the cloud. Recent surveys ([1], [2]) suggest that more and more companies are migrating their applications and services to the cloud instead of owning and maintaining their own IT infrastructures. The main motivators are fast deployment and the pay-only-for-what-you-use nature of the cloud. The high demand for cloud services has led to a drastic increase in the volume of Virtual Machines (VMs) deployed in data centers. For example, using the technique described in [2], we observed that an estimated 360,000 VMs were deployed in 24 hours in the East Coast data center of a major cloud provider.

The high rate of VM deployment in data centers introduces a significant load on the data center network. While the network architectures inside data centers have been designed to accommodate such high network traffic, mainly through installation of expensive, specialized 10GbE switches between racks, those solutions involve significant network cost and contention for limited network resources with application traffic to/from the VMs. In this

thesis, we propose a novel content-based VM scheduling algorithm that can significantly reduce the network traffic associated with transfer of VMs from storage racks to host racks in a cloud data center. Specifically, our algorithm takes advantage of the fact that different VM disk images share many common pages, especially if they use the same operating system and operating system version. While cloud providers typically provide a wide choice of VM images, and users can also bring their own VM images, the same operating systems and operating system versions are often used by multiple cloud users, resulting in many common pages. Furthermore, even though the contents of a VM disk change after the VM has been deployed and is in use, they still retain most of their similarity to the base image from which they were deployed. We used that characteristic of VMs to design our content-based scheduling algorithm. When deploying a VM, we search for potential hosts that have VMs that are similar in content to the VM being scheduled. Then, we select the host that has the VM with the highest number of disk blocks that are identical to ones in the VM being scheduled. Once we have chosen that host node, we calculate the difference between the new VM and the VMs residing at the host; then, we transfer only the difference to the destination host. Finally, at the destination host, we can reconstruct the new VM from the difference that was transferred and the contents of local VMs. Our experimental results show that this algorithm can result in a reduction of over 70% in the amount of data transfer associated with deployment of VM images. The savings are significant enough to have implications for data center network design and the network congestion observed by VMs running on a cloud.

Alongside the rapid rise of computing in the form of cloud services, we are also seeing mobile devices become the primary personal computing platform for hundreds of millions of consumers. In 2012, [3] reported that over 1 out of every 5 U.S. adults owned a tablet and 44% of U.S. adults owned a smart phone, with the numbers still increasing. Individuals often own and use multiple types of devices (e.g., smart phones, tablets, and laptops) not just to consume the same content (e.g., watch movies, listen to music), but also to produce and modify personal content such as pictures, videos, and documents. Often, content must be backed up to protect against device loss or theft, and its access must be managed across all of a user's devices. The best approach for backing up and managing users' content on multiple mobile devices is to use some sort of online storage system. Unfortunately, data caps imposed by cellular providers, storage limitations of cloud storage services, and many other factors make synchronization of users' mobile content to online storage very expensive.

In this thesis, we propose an approach that uses techniques similar to those deployed in our VM scheduling algorithm to identify content similarity among a user's multiple devices to lower the cost of content synchronization and backup. Individuals with multiple mobile

devices not only tent to have high levels of content similarity across their devices (sometimes without their even being aware of it), but also have access to different network interfaces (e.g., WiFi, cellular, Bluetooth) at different times of day. We leverage content similarity across devices to reduce data transfer requirements, and exploit different network connections and different cloud and personal storage services to provide users with cost-effective personal cloud storage for mobile devices.

The rest of this thesis is organized as follows. Chapter 2 explores related work in depth, focusing on previous studies and applications of VM content similarity, and cloud storage for mobile devices. In Chapter 3, we present our content-based scheduling algorithm for VMs in data centers, and in Chapter 4, we present our context-aware personal storage cloud service for mobile devices. Our study on content similarity between virtual disk images is presented in Chapter 5. We measure performance of our content-based scheduling algorithm through simulation in Chapter 6. We conclude with Chapter 7.

CHAPTER 2

RELATED WORK

In this thesis, we present our content-based scheduling algorithms for VM deployments in data centers. In this chapter, we describe work of other researchers who have also used content similarity between VMs in designing various systems (VM live migration, deduplicated storage, and memory-sharing among VMs). In Section 2.1, we will describe that other work in detail, and how it relates to our content-based scheduling algorithms.

In this thesis, we also present our context-aware personal storage cloud service for mobile devices. In Section 2.2, we describe currently available cloud storage and backup services for mobile devices. We also discuss what features the existing services are lacking, and how we intend to address those shortcomings in our personal cloud storage service.

2.1 Applications of VM Content Similarity

In Chapter 3, we will present our content-based scheduling algorithms for VM deployments in data centers. In our algorithms, we take advantage of the content similarity property of virtual disk images in deciding where VMs are scheduled during their deployments in data centers. Others have also exploited that property of VMs in designing many other systems. In this section, we describe some of those systems and how some of the results from others' work aided our work.

An extensive evaluation of different sets of virtual machine disk images was done in [4] to test the effectiveness of deduplication for storing VM disk images. That paper shows promising results regarding content similarity among virtual disk images.

A number of research projects have applied deduplication of identical pages among a group of related VMs being deployed [5] or migrated [6] from one source node to a specified destination node. Those projects have taken the destination as given. However, in our algorithms, we intend to find the best destination node during the scheduling phase of VM deployments to maximize the data transfer savings. A special case of content similarity is considered in [7], which describes a process in which a VM is repeatedly migrated back and forth between two nodes. The pages of the migrated VM are stored at the original source

node, and when the VM is migrated back to this node, only the pages that were modified need to be migrated back. Effectiveness of this system suggests that contents of the virtual disk images do not change much while the VMs are running. This means that content-based scheduling algorithms we are presenting can be successful in finding content similarity between different VMs, even if those VMs have been running for long periods of time.

In the procedure described in [8], replicas of the VMs are stored in different nodes in order to speed up live migration of VMs. When a VM needs to be live-migrated, it is migrated to another node, where a replica of that VM is stored. In order to lower storage costs associated with storage of VM replicas, the authors use deduplication. Placement of VM replicas is based on content similarity between VMs; replicas are placed at nodes where storage savings can be maximized. In that project, content similarity was exploited to improve storage efficiency. In our work, we intend to use content similarity to improve network utilization when transferring virtual disk images between racks in the data centers during VM deployments.

The fact that many VM instances share many common chunks or pages is utilized to speed up VM deployment and reduce the workload at the storage nodes in [9]. The strategy has compute nodes act as peers in a VDN (Virtual machine image Distribution Network), where a VM being deployed can be constructed from chunks being pulled from different compute nodes. However, [9] does not consider using content similarity in the scheduling decision.

A memory-sharing-aware placement of VMs in a data center was presented in [10]. Many data center virtualization solutions use content-based page sharing to consolidate the server's RAM resources. [11, 12, 13] studied maximization of page sharing in virtualized systems. In [10], running VMs with similar memory content are live-migrated to the same hosts. The costs associated with live migration may diminish the benefits of memory sharing in a data center. VM disk images with high content similarity share many common libraries and applications. Therefore, our proposed content-based scheduling of VMs can lead to high memory sharing without live migration of running VMs.

2.2 Cloud Storage and Backup for Mobile Devices

Later, in Chapter 4, we will present our context-aware personal storage cloud service for mobile devices. In our personal cloud storage service, we attempt to make backup and synchronization of mobile data seamless and cost-effective. In this section, we describe similar services that are currently available for mobile devices and the similarities and differences between them and our service.

There are several services, such as Apple iCloud, Google drive, and Dropbox, that synchronize mobile device content with cloud storage. Pricing varies; e.g., iCloud provides an initial 5GB for free with $20/year for each additional 10GB [14]. Applications such as MyBackup [15] and BullGuard [16] let users back up content selected based on data type. The user has to select manually the types of data to back up, and the files are backed up either on the SD card on the same mobile device or on online servers. In our personal cloud storage service, we intend to support backing up on other mobile devices and optimization of storage and networking costs, which are not supported in the above services, to our knowledge.

There has been a lot of work on general peer-to-peer (P2P) and cloud storage systems. Venti [17] is a network archival storage system that provides a write-once archival repository that can be shared by multiple client machines. A P2P backup solution is presented in [18] that combines Venti with DHT; much as in Venti, each unique block is stored only once, even if the block is shared by multiple users. OceanStore [19] and CFS [20] are other P2P archival storage systems that use distributed hash tables. Pastiche [21] provides a P2P backup system in which peers utilize free disk space on other machines. Nodes minimize storage overhead by selecting peers that share a significant amount of data. Finally, [22] presents a P2P storage solution designed specifically for dynamic collections of power-constrained mobile devices on personal area networks (PANs), and its utility functions consider factors such as available power and storage as well as pair-wise locality. To our knowledge, none of these systems utilize predicted network and storage connectivity to optimize costs, which we intend to do in our proposed service.

A number of distributed file systems support file access from mobile devices as well. BlueFS [23] is designed to reduce energy usage and efficiently integrate portable storage. The Coda file system [24] is another distributed file system designed for mobile computing. BlueFS borrows several techniques from Coda and attempts to improve the power consumption. In our storage system, we can take advantage of the fact that it is maintaining files for one user (personal cloud storage), we can avoid many of the complexities of general-purpose file systems, such as the need to maintain consistency across concurrent accesses.

In Chapter 3, we will present our novel content-based VM scheduling algorithms for cloud data centers. We will describe how those algorithms can lead to significant reductions in network traffic associated with transfer of VMs between racks during the deployment phase. In Chapter 4, we will present our context-aware personal cloud storage service, which allows consumers to back up and manage content across their multiple mobile devices. We use some of the same techniques employed in our content-based scheduling algorithms (see Chapter 3) to optimize the storage and networking costs of our personal cloud storage service. We also discuss our analysis of how effective such a proposed personal cloud storage service can be.

To evaluate the effectiveness of our content-based scheduling algorithms from Chapter 3, we performed an extensive content similarity study between virtual disk images, discussed in Chapter 5. Then, using our findings from Chapter 5, we ran simulations of VM deployments in data centers. Those simulations and their results are described in Chapter 6.

CHAPTER 3

CONTENT-BASED SCHEDULING OF VMS

3.1 Motivation

The number of virtual machines (VMs) deployed in a large cloud data center each day can be very large, and their deployment introduces a significant load on the data center network. In only one of Amazon's many data centers (the East Coast data center), we observed that an estimated 360,000 VMs were deployed in 24 hours (using the technique described in [2]). In typical data centers, VM disk images are stored in specialized storage racks and then transferred to compute nodes in other racks when a VM based on the image is deployed. The VM disk image sizes typically range from around 1 GB to tens of GBs. Transfer of such large numbers of such large VM disk images inside a data center introduces a significant amount of network traffic between racks. Usually, data centers address this problem by installing specialized 10GbE switches between racks. That doesn't always address the whole problem, because such switches add significant cost to data centers. If we can reduce the network traffic associated with transfer of VM disk images, then there is less contention for application traffic to/from the VMs.

Our proposed content-based VM scheduling algorithm aims to do just that. In a later chapter, we show the findings of our study of content similarity between various VM disk images. Our findings show that there is significant content overlap between VM images that share the same OSes and/or OS version numbers. That led us to design the content-based scheduling algorithm we describe in this chapter.

While researchers have utilized the observation of identical pages in different VM images in the past to optimize VM deployment or live migration [8, 5, 6, 7], our algorithm is, to our knowledge, the first one to utilize content similarity in VM disk images to optimize VM scheduling in a data center.

3.2 Background: Scheduling of VMs in Data Centers

In a typical Infrastructure as a Service (IaaS) deployment, a pool of VM disk images is stored in storage nodes. The images are templates for virtual machine file systems. VM instances are instantiated from these images at compute nodes. To deploy a VM instance, a user selects an image and instance type. An instance type typically specifies physical resources (such as CPU or RAM) that will be allocated for the instance once deployed. Once the user has specified the image he or she wants to deploy, a scheduling algorithm selects a compute node and copies the image from the storage node to the local storage of the compute node. Once copied, the VM can be booted up at the compute node.

The part of this process in which we are interested is the scheduling of VMs. The methods of VM scheduling algorithms used by major cloud service providers are proprietary. Hence, to explain VM scheduling in data centers, we will refer to the scheduling algorithm implemented in OpenStack [25]. OpenStack is open-source cloud software supported by thousands of developers, researchers, and the open-source community, in addition to hundreds of leading companies. The default scheduling implementation in OpenStack is the *filter* scheduler, which consists of two phases: *filtering* and *weighting*.

The task of the *filtering* phase is to eliminate the compute nodes without sufficient resources (such as CPU or RAM) to host the new VM. The *weighting* phase assigns weights to the remaining compute nodes based on the states of the compute nodes and properties of the VM being scheduled. Its purpose is to select the most appropriate host for the VM being scheduled. For example, it would not be optimal to schedule a simple VM with low resource requirements on a high-performance host. The weights can incorporate load-balancing policies in the data center, utilization of nodes, and how well the available resources of the compute nodes match up with the VM resource requirements.

3.3 Content-Based Scheduling Algorithm Architecture

Our scheduling algorithms were designed with the goal of lowering the amount of data transferred between racks in the data center when VM disk images are being copied to the host nodes. We achieved that goal through the co-location of VMs with high content similarity at the same hosts.

The architecture of our scheduler is as follows. There are management nodes that are separate from compute and the storage nodes in the data center that hosts the scheduler. Requests to deploy VMs are sent to the scheduler, which selects an appropriate compute node for each VM. The scheduler stores fingerprints for all of the VM disk images from

```
1: function SELECT HOST(VM_N)
2:     Filter nodes based on available resources
3:     Filter nodes if VM_N OS different from node OS
4:     N ← n randomly selected nodes from filtered nodes
5:     maxSimilarity = −1
6:     for node ∈ N do
7:         for VM ∈ node do
8:             similarity = calcSimilarity(VM, VM_N)
9:             if similarity > maxSimilarity then
10:                maxSimilarity = similarity
11:                selectedNode = node
12:                selectedNodeVM = VM
13:            end if
14:        end for
15:    end for
16:    return (selectedNode, selectedNodeVM)
17: end function
```

Figure 3.1: Dedicated node scheduling algorithm

the image library. It also stores fingerprints for every running VM in the data center. For a running VM, the scheduler maintains a mapping between the VM's fingerprint and the compute node hosting the VM. All the fingerprints are stored on the management nodes. When VMs terminate, their fingerprints are removed from management nodes. Fingerprints are described in Section 3.4.2; they are used to estimate content similarity between two VM disk images.

We have designed two different content-based scheduling algorithms.

Dedicated nodes algorithm: In this algorithm, each compute node is dedicated to hosting VMs with the same OS. For example, if a compute node is hosting a Ubuntu VM, then all the VMs hosted on the node will be Ubuntu VMs. We do not require the version numbers of the VMs to be the same on a compute node. Nodes get assigned specific OSes dynamically, as follows. When some VM with OS_X is being scheduled, if there are no nodes dedicated to OS_X or if all the nodes dedicated to OS_X are full, then the VM is assigned to a node that has no VMs running on it. As a result, that node becomes a dedicated node for OS_X. When all the running VMs on a certain node are terminated, that node is no longer dedicated to any OS.

In lines 2 and 3 in Figure 3.1, the scheduler eliminates the nodes that are not dedicated to the same OS as the new VM, and the nodes that do not have enough available resources to host the new VM. What remains are nodes that are dedicated to the new VM's OS. Next, the scheduler selects n nodes randomly from the remaining nodes. In Chapter 6, we will

discuss how the different values of n affect the performance of the algorithm. The reason for randomly selecting the nodes is to balance the load in the data center. Then, the scheduler does content comparison between the new VM and all the running VMs at the n selected nodes (lines 5–15 in Figure 3.1). The new VM is assigned to the node hosting the VM with the highest content similarity to the new VM. The algorithm returns the selected node and the VM on that node with the highest content similarity, and that VM is used during transfer of the new VM to that node.

Greedy algorithm: In this algorithm, we do not require the host nodes to be dedicated to any one OS; rather, the nodes can host VMs with any combination of OSes. As in the above algorithm, in the first step, the scheduler filters out all the nodes that do not have enough resources available to host the new VM. Then, it iterates over all the remaining nodes, computing content similarity between the new VM and all the VMs running on the host nodes. It selects the node hosting the VM with the highest content similarity. Compared to the dedicated nodes scheduler, this approach is more computationally intensive, because many more nodes are evaluated to find the highest content similarities. The greedy algorithm can be adjusted to limit the number of nodes it inspects, so that it does not need to inspect all the nodes in the system. In our simulations (see Chapter 6), we studied how well the greedy algorithm works even when all the nodes are inspected to identify the maximal content similarity.

3.4 Algorithm Implementation Details

In this section, we will describe implementation details of the two main components of our scheduling algorithm: *VM disk image fingerprints* and the *VM disk image transfer algorithm*. First, we describe the basic characteristics of Bloom filters.

3.4.1 Bloom Filter

A Bloom filter is a space-efficient randomized data structure used to represent a set in order to support membership queries. Small Bloom filters can be used to represent large sets. Bloom filters achieve such space efficiency at the cost of false positives, but the space savings often outweigh this drawback. It is possible to bring the false positive rates very low while still maintaining space efficiency. Bloom filters were introduced in the 1970s by Burton Bloom [26] and have since been used in a range of different areas.

The basic algorithm for generating Bloom filters is the following. A bit array of m bits is

used to represent the Bloom filter. For an empty Bloom filter, all entries of the bit array are set to 0. The algorithm also requires k different hash functions. Each of them has functions that should hash to one of the bit array values (0 to m) with uniform distribution. To add elements to the Bloom filter, k different hash values of the element are generated using the k hash functions. Then, entries of the bit array corresponding to the hash values are set to 1. To check whether an element exists in the Bloom filter, k different hash values of the element are generated using the k hash functions again, and are checked to ensure that all the bit array entries at those hash values are set to 1. If any of the entries aren't set to 1, then with 100% probability, we report that the element is not in the Bloom filter. Otherwise, with some probability less than 100%, the element is reported to be in the Bloom filter.

The probability of false positives depends on the values of k, m, and total number of elements n in the Bloom filter. The relations are described below. For a given m and n, the value of the number of hash functions (k) that minimizes the probability of false positives is [27]:

$$k = \frac{m}{n} \ln 2 \tag{3.1}$$

Assuming that an optimal value of k from Equation 3.1 is used, the probability of false positive is [27]:

$$p = \left(1 - e^{-(m/n \ln 2)n/m}\right)^{(m/n \ln 2)} \tag{3.2}$$

which can be simplified to:

$$\ln p = -\frac{m}{n} (\ln 2)^2 \tag{3.3}$$

The biggest advantage of Bloom filters over other data structures is that depending on the application, the false positive probability, the size of the Bloom filter, and the number of hash functions used can be controlled to fit the application's specific needs.

3.4.2 VM Disk Image Fingerprints

We implemented the VM disk image fingerprints using Bloom filters [28]. Figure 3.2 shows the algorithm for generating fingerprints using a Bloom filter. Each fingerprint represents the contents of one VM disk image. In Figure 3.2, a fingerprint is represented by a bit array of size m. We also define k different hash functions. In our implementation, we use common hash functions, such as `md5`, `sha1`, and `sha256`. To get k different hash functions, we generate multiples of `md5`, `sha1`, and `sha256` using different salt values.

```
1: function GENERATE FINGERPRINT(diskImage)
2:     fingerprint ← bitArray(m)
3:     fingerprint.setAll(0)
4:     hf_1, ..., hf_k ← k different hash functions
5:     while block = diskImage.read(4096) do
6:         for i = 1 → k do
7:             arrayIndex = hf_i(block) % m
8:             fingerprint[arrayIndex] = 1
9:         end for
10:    end while
11:    return fingerprint
12: end function
```

Figure 3.2: Fingerprint generation for VM disk images

Starting on line 5 of the algorithm, we read the contents of the disk image in 4096B chunks. The whole VM disk image is split into 4096B fixed-size disk blocks, and each 4096B disk block represents an element in the set. For each disk block, we generate k different hash values using the hash functions, and set to 1 the entries of the fingerprint bit array corresponding to the hash values. The algorithm finishes when all the disk blocks of the disk image have been added to the fingerprint. Since each VM disk image is represented as a set, duplicate disk blocks of the image are ignored in the fingerprint.

One of the main reasons we selected Bloom filters, besides space efficiency, is that they allow for easy and efficient calculation of the intersections between two sets [28] as follows:

$$\frac{1}{m}\left(1-\frac{1}{m}\right)^{-k|S_1\cap S_2|} \approx \frac{Z_1+Z_2-Z_{12}}{Z_1 Z_2} \tag{3.4}$$

Here, k and m are the same as in Figure 3.2. S_1 and S_2 represent the two sets; Z_1 and Z_2 represent the number of 0s in the bit arrays for S_1 and S_2, respectively. Finally, Z_{12} represents the number of 0s in the inner product of the two bit arrays. We solve the equation for $\mid S_1 \cap S_2 \mid$ to calculate the approximate size of the intersection of S_1 and S_2.

We use Equation 3.4 to calculate content similarity between two fingerprints representing two VM disk images. Solving for $\mid S_1 \cap S_2 \mid$ in Equation 3.4 gives an estimate of the number of 4096B disk blocks that are identical between the two VM disk images. We ran VM comparisons using fingerprints with smaller block sizes (512B, 1024B), but the accuracy of the content similarity calculation increased very little. Also, using 4096B disk blocks for fingerprints resulted in smaller-sized fingerprints; therefore, we chose 4096B disk blocks for the fingerprints.

3.4.3 Transfer Algorithm

Figure 3.3 shows the algorithm for transferring VM disk images from storage nodes to host nodes once the scheduler has selected a host node. It is similar to the `rsync` algorithm, and has four phases. Let VM_N be the VM that is being transferred, and VM_L be the VM that is running on the destination node. In the first phase, the source (storage) generates the md5 hash values for each of the 4KB disk blocks that make up the VM_N. Then, these hash values are sent to the destination (compute) node. In the second phase, once the destination node has received the list of hash values from the source node, it also calculates the md5 hash values for the disk blocks of the local VM_L. Next, the destination node makes a list of the VM_N's hash values that do not appear in VM_L. They correspond to the VM_N's disk blocks that are not in VM_L. The destination node requests the missing disk blocks from the source node. In phases 3 and 4, the source node sends the missing blocks to the destination node, and the destination node reconstructs the VM_N using the blocks from VM_L and the missing blocks received from the source node.

The size of each md5 hash value is 16B, and each hash value represents a 4KB disk block; therefore, the list of hash values in phases 1 and 2 is much smaller than the VM being deployed.

1: **Phase 1:** Source node
2: Let VM_N be the VM disk image being transferred
3: **while** $block \leftarrow VM_N.read(4096)$ **do**
4: $\quad blockHash \leftarrow md5(block)$
5: $\quad hashList.append(blockHash)$
6: **end while**
7: Send $hashList$ to destination node
8:
9: **Phase 2:** Destination node
10: Receive $hashList$ from source node
11: Let VM_L be the local VM with highest content similarity
12: **while** $localBlock \leftarrow VM_L.read(4096)$ **do**
13: $\quad blockHash \leftarrow md5(localBlock)$
14: $\quad localHashList.append(blockHash)$
15: **end while**
16: **for** $blockHash \in hashList$ **do**
17: $\quad$ **if** $blockHash \notin localHashList$ **then**
18: $\quad\quad missingBlocksList.appendP(blockHash)$
19: $\quad$ **end if**
20: **end for**
21: Send $missingBlocksList$ to source node
22:
23: **Phase 3:** Source node
24: Receive $missingBlocksList$ from destination node
25: **for** $blockHash \in missingBlocksList$ **do**
26: $\quad offset \leftarrow hashList.indexOf(blockHash) \cdot 4096$
27: $\quad block \leftarrow VM_N.read(offset)$
28: $\quad missingBlocksKeyValue[blockHash] \leftarrow block$
29: **end for**
30: Send $(missingBlocksKeyValue,\ md5(VM_N))$ to destination node
31:
32: **Phase 4:** Destination node
33: Receive $(missingBlocksKeyValue,\ md5(VM_N))$ from source
34: Combine local blocks from VM_L and received blocks from VM_N to reconstruct VM_N locally
35: Generate $md5$ hash for local VM_N and verify it matches $md5$ of VM_N received from source node

Figure 3.3: VM disk image transfer algorithm

CHAPTER 4

CONTEXT-AWARE PERSONAL STORAGE

In Chapter 3, we saw how we can utilize specific characteristics of VM disk images to lower network traffic associated with transfer of VMs inside data centers. Specifically, we used Bloom filters to identify similar content across VM images. In this chapter, we use same techniques to lower the cost of backup and synchronization of an individual's content across his/her multiple mobile devices. Mobile devices today not only face high network costs, like data centers do, but face high storage costs as well. We use Bloom filters to identify similar content across a user's multiple devices and use the results to lower costs associated with network and storage for our mobile backup and synchronization solution.

4.1 Background: Cloud Storage Services for Mobile Devices

Mobile devices have become the primary computing platform for many consumers. Today, individuals often own and use multiple types of devices (e.g., smart phones, tablets, and laptops) not just to consume the same content (e.g., watch movies, listen to music), but also to produce and modify personal content such as pictures, videos, and documents. Often, content must be backed up to protect against device loss or theft, and its access must be managed across all of a user's devices. Unfortunately, users often exceed the limited storage capacity of their mobile devices. Therefore, it is often impractical to maintain a full copy of all a user's data on all of his or her devices.

To address the problem of multi-device content management, two primary approaches have been proposed. Either users must partition and replicate their content across their different devices manually, or they must use some form of cloud storage to store a master copy of the content and synchronize their devices to this cloud store. However, neither of those approaches is completely satisfactory. Manual partitioning and management of content places a high burden on users, forcing them to track where different pieces of content are. Tracking becomes even more onerous if some of the content has been modified on some devices but not on others. Historically, while users could often simplify matters by relying on a single *master copy* of their data on a single device (often their home desktop or laptop)

and treating their mobile devices as caches, the prevalence of mobile-device-only households means that it is increasingly difficult to do so.

Cloud storage services such as Dropbox, Google Drive, and Box are the other alternatives to multi-device management that have arisen to fill the void left by the absence of a single home desktop where all the user's content is stored. However, today's cloud-based synchronization is an all-or-nothing proposition; most storage providers (with few exceptions, like iTunes Match) expect a master copy of all data to reside in the cloud and either synchronize all of it with each endpoint, or expect the user to manually manage what data are copied to each device. That model means that either the users must store all their data in the cloud (an expensive proposition depending on how much data they have), or forgo the convenience of an automatically managed and synchronized storage system. Furthermore, cloud storage applications typically ignore the cost of network access completely, forcing the user to quickly use up expensive cellular data plan quotas in synchronizing their data.

In this thesis, we present an alternative approach: an intelligent *personal storage cloud* with the following characteristics:

1) Utilizes *all* of the storage available to users on their mobile devices, cloud storage services, and home servers in a cost-sensitive way. Provides the user with a global namespace, but partitions and replicates user data across the different storage options based on cost, capacity, and usage patterns. In doing so, preferentially leverages storage for which the user has already paid, e.g., storage on the user's mobile devices or a home server, over storage that will incur additional incremental costs (e.g., cloud storage that exceeds free initial quotas). Also leverages *usage context*, i.e., which files are accessed when and on which devices, to optimize placement of content. Specifically, evicts less frequently used file blocks from a device if they are already stored on another storage node, while leaving the metadata needed to efficiently locate the content if the user desires to access it.

2) Is *network-context-aware* in recognizing that different storage nodes (device, cloud storage, home server) are accessible through different network interfaces (e.g., WiFi, cellular, Bluetooth) with differing cost profiles at different times of the day. Uses these different network connections in a cost-effective manner by prioritizing storage nodes that can be accessed cheaply at a given time of day.

3) Leverages content similarity across devices to reduce data transfer requirements, and uses a cost model that reflects the combined effects of storage and network costs, as well as content similarity opportunities, to make optimal content placement decisions. Recent surveys [3], [29] suggest that many U.S. households own multiple smart phones, tablets, and other mobile devices. We postulate that there is a large amount of similar content among each household's devices. For example, devices owned by the same person or household may

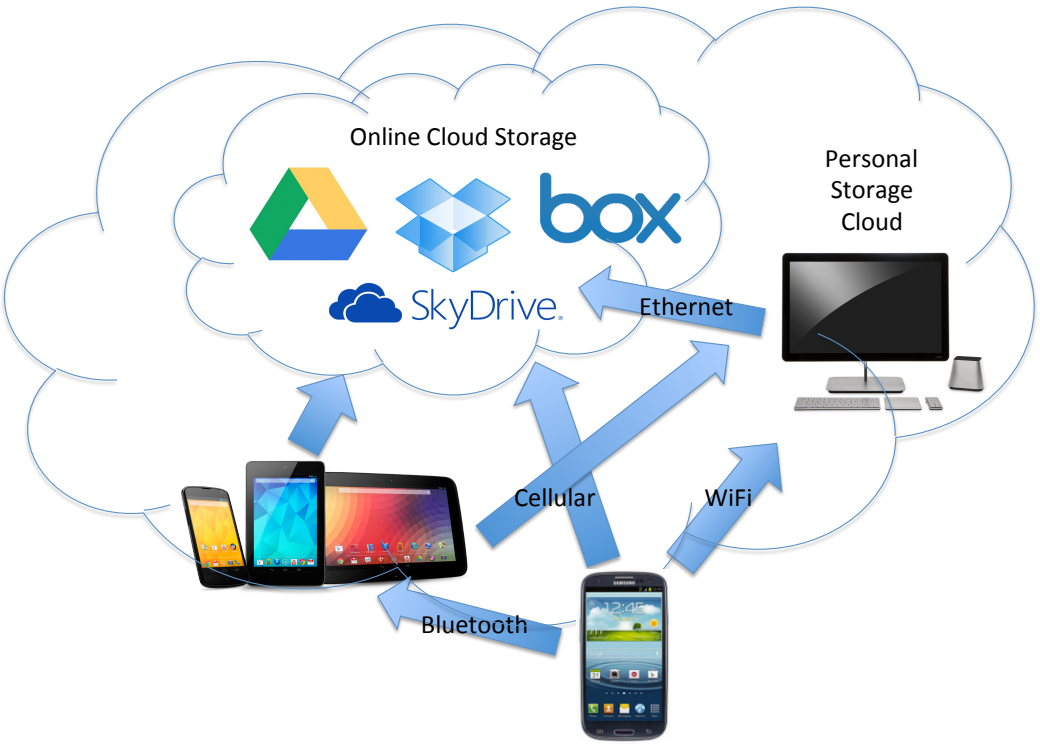

Figure 4.1: System overview

share applications, user-generated content such as photos and videos, and consumption media such as songs and movies. If that content similarity can be efficiently and effectively tracked in an online manner, it can provide an excellent source of ready-made data redundancy that could not only reduce the storage needs of device backups, but also reduce the amount of network bandwidth needed for such functions.

Figure 4.1 captures an overall picture of our system.

4.2 Challenges and Motivation

In the past few years, we have seen exponential growth of smart phones and tablets in the mobile market. For example, according to [29], there were 500 million Android devices activated as of September 2012, and during the past year, there were an average of 1.3 million new Android device activations every day. In addition, the popularity of tablets in recent years has also contributed to rapid growth of mobile devices in households. Last year, [3] reported that more than 1 out of 5 U.S. adults owned a tablet, and almost half of U.S. adults owned smart phones, with the trend going upwards. These figures suggest that many households or even individuals own multiple of these mobile devices. The plurality of mobile

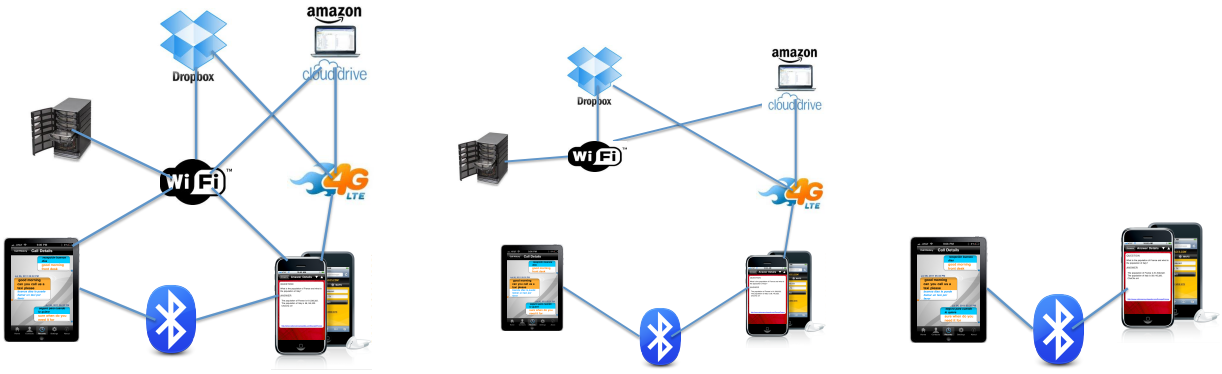

Figure 4.2: Usage scenarios (at home, on the road, international travel)

devices introduces both new requirements and opportunities. Specifically, users want to access content from multiple devices seamlessly and move to new devices easily; in addition, we also want to find ways to use the additional devices (as well as online cloud storage and home PCs/storage) as backup storage and to virtually expand the limited storage capacity of small mobile devices.

Online cloud storage services, like mobile devices, have also proliferated over the last few years (e.g., DropBox, iCloud, Box, Google Drive, Amazon S3). While some services support storage of specific types of information online (e.g., calendar, contacts, email), others provide general-purpose file storage. Storing files online in cloud storage makes accessing of content from multiple devices easy, provides backup, and allows files to be removed from mobile devices and stored online only. While those services are useful building blocks for our context-aware personal cloud storage service, none of them alone can provide the seamless and cost-effective storage abstraction we present.

Figure 4.2 presents three different sample scenarios in which a user's mobile devices and cloud storage services, including home storage, might be connected using networks with greatly varying performance and cost characteristics. In the "at home" scenario on the left, all mobile devices and storage nodes are accessible through a relatively low-cost and high-performance WiFi network. In addition, there is plentiful local storage available through the user's home server. In the "on the road" scenario in the middle, connectivity is more limited; the smart phone can communicate with cloud storage or a home server only through a cellular network, which may charge based on the amount of data transferred, while the tablet can directly communicate with the smart phone only by using Bluetooth. Finally, the "international travel" scenario on the right shows an extreme situation in which high roaming costs force a user to rely solely on local connectivity options, such as a Bluetooth connection between the devices the user is carrying on the trip. We call such a graph that encodes connectivity among *all* of a user's devices and storage options the *network context graph*. It is time-dependent, and contains vertices and edges that represent storage nodes and

network connections. Given such a graph, we can find times throughout the day when paths with the lowest costs are available, and we may be able to schedule content synchronizations at such times to minimize cost. Alternatively, we may also be able to find the lowest-cost path at any given time in order to choose a target node from several different options.

4.3 Context-Aware Personal Storage Service for Mobile Devices

In this section, we provide the design and architecture of our personal storage cloud service. We assume that each user has one or more mobile devices, as well as access to a number of cloud storage services and/or a personal storage server. Each cloud storage service is associated with a cost utility function, i.e., a mapping between how much storage is used and the cost. Typically, cloud storage services use tiered utility functions, according to which the users pay a fixed cost for "renting" a certain amount of storage, whether they use it or not. The initial tier is often free. We create separate cost function for each individual type of storage node. For example, Amazon S3 storage has a cost function based on the price of storage. Amazon S3 charges for how many GBs a month was used and for each `PUT` and `GET` operation. On the other hand, Dropbox doesn't charge anything for its free tier storage accounts; hence, the cost function associated with this storage node will be based on the amount of available space on the account. For home servers that are accessible only through local area networks, the cost function will be based on availability of that node, i.e., how often the user has access to his/her home servers through local area networks.

We also consider tiered network costs; e.g., cellular network providers often charge based on the total volume of data transferred. Therefore, each device can also be associated with a networking cost utility function that is similar to the storage utility function (e.g., cost for a certain number of bytes transferred). Such a formulation also encompasses the scenario (e.g., for wired networks, or WiFi) in which usage is not volume-based, but incurs a fixed cost irrespective of the amount of data transferred (within some limit). We designed a file placement policy with the goal of minimizing the total cost incurred by the user, including the storage and the network costs.

4.3.1 Network and Storage Context Graph

We assume that the user provides the system with a list of his/her mobile devices, servers, and on-line storage accounts. We model the file placement problem as a graph in which nodes are the storage nodes and the mobile devices, and the edges represent communication

links. Each node has a storage utility cost function that is based on several factors: the availability of the node, the capacity of the node, storage tier price, and I/O price. Each pair of nodes representing mobile devices has edges representing different communication links between these devices (e.g., Bluetooth, LAN, WiFi, or cellular). Similar edges connect mobile device nodes to the storage nodes. Like graph nodes, the edges have utility cost functions that are determined by factors such as price per volume of data transferred, power consumption associated with specific types of communication link, and a cap on the amount of data transferred. The graph is used to calculate the overall costs of moving content from devices to storage nodes.

For each edge in the graph, we build availability profiles for different times of the day by periodically monitoring availability of that edge. To enable that, on the mobile devices, a periodic background task requests network connectivity information once every hour. The resulting edge profiles help us assign probabilities to the availability of specific edges at different times of day. For example, if a user has a home server, we build a profile showing when the user is likely to be on the same local network as the home server. Over time, that type of monitoring allows us to find patterns in users' everyday lives in terms of access to various communication links and storage nodes.

4.3.2 Usage Context Monitoring

In addition to building profiles for communication links to storage nodes, we monitor the file systems on the mobile devices to determine users' file access patterns. A background task on the mobile device monitors file system activity, such as file access, modification, deletion, or creation. Such usage profiles allow us to calculate cost more accurately when assigning contents from mobile devices to storage nodes. For example, files that are frequently modified may be more expensive to store on a cloud storage node vs. the home server if the cloud storage provider charges for I/O (e.g., Amazon S3 charges for `PUT` operations). In addition, the profile can be used to identify infrequently accessed files to determine which file contents to remove from the mobile devices when their local storage gets exhausted, so that they are stored only in one of the cloud storage nodes.

Another important function of the file system monitor is to keep track of content that is identical on multiple devices. When the algorithm assigns content to storage nodes, awareness of content similarity between the local device and other nodes may reduce cost. Each device keeps track of local files in a data structure similar to Bloom filters [28]. Bloom filters allow quick calculation of the intersection of two sets.

4.4 Implementation

In our implementation of our cloud storage system, we maintain mapping between files and the storage locations where those files are replicated and stored. Initially, for all the files in the mobile device, we calculate the cost of storing those files at different storage nodes. The storage nodes can be a user's various cloud storage accounts (e.g., Dropbox, Google Drive, Box), the user's personal home servers, or the user's other mobile devices. The storage nodes are entered and configured by the user. Files are assigned to storage nodes, where they are replicated and periodically synchronized, based on the lowest-cost storage nodes for those files.

At each mobile device, we use Bloom filters to keep track of local files on that device. When we assign files to storage nodes, we use Bloom filters to identify storage nodes that may already have the same files. Those storage nodes lead to lower storage and network costs for those files, because the files don't need to be transferred over expensive network links, and at the storage nodes, we only keep a reference to the existing files, leading to storage savings.

At each of the storage nodes, we store a metadata file that lists file names that are replicated at that site and their hash values. We periodically synchronize the local device and the storage sites. During synchronization, if local files have changed since the last synchronization, then they are updated at the corresponding storage nodes where they are replicated. During synchronization, we query only the metadata files from storage nodes. We compare the hash values of the files from metadata to hashes of local files. If the two hashes differ, then the corresponding files are marked for update.

We implemented this service on the Android platform. To build the availability profiles discussed in Section 4.3.1, we used a combination of `AlarmManager` and `BroadcastReceiver` in Android. They allow us to wake up the mobile device only once an hour to collect network connectivity information, but don't consume any system resources at other times. Also, to monitor file system access patterns on users' mobile devices, we use the `FileObserver` class from Android.

4.5 Performance Analysis

4.5.1 Multi-Device Content Similarity

A unique aspect of our proposed personal storage cloud is the use of content similarity among a user's different devices. That similarity can be exploited to minimize the overall cost. Multiple devices owned by the same person—or household—often have the same movies, pictures, music, or documents stored on them. When the devices are being backed up, identifying and utilizing similar content across devices can greatly improve performance and cost. To analyze the extent of such similarity, we asked several of our colleagues, who owned multiple mobile devices, to volunteer for our analysis. We compared the data on each person's devices to determine how much was the same across the devices. The results are shown in Table 4.1.

Table 4.1 displays the types of devices each user has, and the amount of data stored on each device. The last column lists the amount of data that is identical on the two devices. The results clearly show that content similarity between users' devices varies from user to user. In the cases of users A and B, the amount of data shared between the devices is very large. But in the cases of users C and D, the data overlap between the users' devices seems negligible. We can better understand the results by analyzing how each user uses his or her devices.

User D used his tablet mainly for class and work-related tasks (e.g., taking notes, making documents and slide presentations). He used his smart phone mainly for personal use. That explains the lack of content similarity between this user's devices. User C, on the other hand, barely had any content stored on his smart phone. It can be seen that the total amount of data stored on his smart phone added up to only 2.1GB. To watch movies and listen to music, this user used different online media streaming services (e.g., Netflix, Hulu, Pandora), rather than store media content locally.

For users A and B, most of the shared content was due to movies and music. In addition to using streaming media services, these users stored music and movies for offline consumption.

The above sample of users captures the general behavior of various users who own and use multiple mobile devices. In general, people who rely on offline content (locally stored movies, music, pictures, and so forth), will have greater content similarity among their mobile devices than people who mostly use online media streaming services. In the former case, our context-aware personal storage service will benefit the users greatly by taking advantage of content similarity among their devices to minimize both network and storage costs. On the other hand, our service will also provide low-cost backup storage to people who don't have

Table 4.1: Mobile Device Content Similarity

User	Device 1 Type & Size	Device 2 Type & Size	Shared Size
User A	laptop – 25GB	laptop – 135GB	10.9GB
User B	phone – 4.5GB	laptop – 65GB	4.3GB
User C	phone – 2.1GB	tablet – 10.2GB	41MB
User D	phone – 8.1GB	tablet – 16.4GB	102MB

much offline content, because the amount of data they have to back up is small to begin with.

4.5.2 Network and Storage Context Graph Analysis

In addition to performing content similarity analysis, we observed two users' network and storage context for over a week learning what networks the users' devices could access, and when their devices were co-located. In each device, a background task would fire once every hour and record network information and the GPS location of the device. One of the users had a WiFi-only Nexus 7 tablet and had an FTP server on his home network. The other user had a WiFi-only Nexus 7 tablet and an Android 3G phone. Both users had WiFi networks at home and at the office.

The network context is presented in Table 4.2. For each of the six listed types of network-device connectivity, the table gives the fraction of days for each time window during which that connectivity was observed (averaged over durations of the logs). The "T1 W" column shows the fraction of days (number of days the connection was observed divided by total number of days the logs were taken for) Tablet 1 (belonging to the user who had both a tablet and an Android phone) had WiFi access; this tablet had WiFi access less than half of the time for most time intervals. That user's Android phone was connected to a WiFi network ("P1 W" column) more than to a cellular network ("P1 Cell" column) during the night, but at certain times of the day (11:00 a.m. to 7:00 p.m.), the phone was connected to cellular network coverage more than 50% of the time. The numbers from the table show what fraction of time the devices were connected to the corresponding networks, but that doesn't mean the networks were unavailable rest of the time. For example, we observe that the Android phone was connected to WiFi 66% of the time for the 12:00 to 1:00 a.m. window ("P1 W" column, "12 - 1AM" row), and was connected to a cellular network 33% of the time during the same window ("P1 Cell" column, "12 - 1AM" row). That doesn't mean that cellular network coverage was only available 33% of the time at midnight for this user, but that during this time, the phone was connected to WiFi instead of a cellular network 66% of

Table 4.2: Hourly Network Profiles

Time	T1 W	P1 W	P1 Cell	T1 - P1 B	T2 W	T2 FTP
12 - 1AM	0.44	0.66	0.33	0.92	0.77	0.85
1 - 2AM	0.44	0.66	0.33	0.92	0.86	0.79
2 - 3AM	0.44	0.66	0.33	0.92	0.86	0.79
3 - 4AM	0.44	0.66	0.33	0.92	0.86	0.85
4 - 5AM	0.44	0.75	0.25	0.95	0.86	0.92
5 - 6AM	0.44	0.75	0.25	0.95	0.86	0.92
6 - 7AM	0.44	0.25	0.75	0.71	0.86	0.92
7 - 8AM	0.50	0.50	0.50	0.63	0.86	0.92
8 - 9AM	0.38	0.90	0.10	0.63	0.86	0.92
9 - 10AM	0.33	0.69	0.31	0.63	0.87	0.73
10 - 11AM	0.38	0.58	0.42	0.60	0.73	0.87
11 - 12PM	0.38	0.42	0.58	0.47	0.71	0.64
12 - 1PM	0.38	0.36	0.64	0.31	0.61	0.69
1 - 2PM	0.38	0.58	0.42	0.12	0.75	0.36
2 - 3PM	0.62	0.29	0.71	0.12	0.77	0.54
3 - 4PM	0.38	0.25	0.75	0.12	0.71	0.50
4 - 5PM	0.38	0.36	0.64	0.12	0.71	0.53
5 - 6PM	0.38	0.42	0.50	0.12	0.85	0.38
6 - 7PM	0.38	0.42	0.50	0.12	0.77	0.38
7 - 8PM	0.44	0.64	0.46	0.43	0.77	0.38
8 - 9PM	0.44	0.62	0.38	0.64	0.77	0.46
9 - 10PM	0.44	0.62	0.38	0.83	0.71	0.57
10 - 11PM	0.44	0.54	0.46	0.83	0.71	0.50
11 - 12AM	0.44	0.66	0.33	0.92	0.79	0.71

the time. (Cellular network coverage could have been available more than 33% of the time, but most devices prefer to connect to WiFi networks over cellular networks, when possible).

The "T1 - P1 B" column shows (for each time interval) the fraction of days during which this user's Android phone and tablet were within Bluetooth range of each other. The user did not take his tablet to the office during the day, when he was at work. Thus, the tablet and the phone were within Bluetooth range of each other most of the time during the night, but, during the day, the fraction of time they were within Bluetooth range was very low.

The "T2 W" column shows the WiFi connectivity of Tablet 2 (belonging to the user with no other mobile devices). This user carried his tablet with him most of the time. The tablet had over 70% WiFi access during all time intervals except 12:00 to 1:00 p.m. The reduced WiFi access over the noon hour may reflect the user's tendency to step out for lunch. Finally, the "T2 FTP" column shows what fraction of the time the user's tablet was on the same WiFi network as the user's home server. The fact that this user often worked from home is reflected by the results in the last column.

The results show that network connectivity and access to other devices can vary greatly throughout the day, and that careful selection of when and where to back up (another co-located device or cloud) will lead to substantial cost savings if one waits for a time when cheap network connectivity is available (WiFi or Bluetooth).

4.5.3 Results Summary

By combining the results from Sections 4.5.1 and 4.5.2, we can estimate how much savings in cloud storage and cellular usage charges our personal storage cloud could deliver. E.g., assume a user has a smart phone with 32GB of storage, a tablet with 128GB of storage, and a home server with a 1TB disk. The user could naively choose to back up the mobile devices to on-line cloud storage (160GB total). However, assume a content similarity of 40%. In that case, 51GB of the data on the tablet is already present on the home server, and 25GB of the data on the smart phone is already included on the tablet (or home server). Thus, only the remaining 74GB would need to be backed up to cloud storage. Furthermore, if we actively use the home server to back up both of the mobile devices when the devices can access the home wireless LAN, and use the tablet to back up the smart phone when away from home but co-located and reachable by Bluetooth, the demand for on-line cloud storage can be reduced to the size of the new content created on the mobile devices that is so critical that it cannot wait to be backed up on the home server (or the other mobile device). This storage requirement would likely be satisfied by free first-tier on-line cloud storage services.

CHAPTER 5

VM CONTENT SIMILARITY STUDY

In this chapter, we present our findings on content similarity between VM disk images. We mainly studied VMware images, and some Amazon machine images (AMIs). We collected about 50 different VM images (Linux- and Unix-based OSs) from [30] and 10 images from [31]; some of them are listed in Table 5.1.

We used the Bloom-filter-based fingerprints (described in Section 3.2) in all of our comparisons. We generated a fingerprint for each VM disk image. To calculate content similarity between two images, we calculated the intersection of the corresponding fingerprints. In our experiments, content similarity estimates based on the fingerprints were within 1% of the actual content similarities. VM disk images often have duplicate blocks, but the fingerprints allow us to determine the total size of the VM disk image without the duplicate blocks. In Table 5.1, in the second column, we first show the size of each stored VM disk image on the file system (size with duplicate blocks), and then, in parentheses, its size without duplicate blocks.

5.1 Base Virtual Disk Image Comparison

In this section, we discuss the content similarity results between the base VM disk images. Results of our comparisons are shown in Table 5.2. We found that VMs with different OSes have no content in common; therefore, such pairs are not displayed in the table. In Table 5.2, we show the comparison between base VM disk images with the same OS but different version numbers. The sizes of the VM disk images are shown in Table 5.1, and the last column of Table 5.2 shows the amount of data that is common to *VM 1* and *VM 2*. The percentages of content similarity (columns 2 and 4) between images are calculated by comparing the **Shared** column in Table II against the **Image Size** column in Table I. For example, consider the first row of Table 5.2 as an example. The size of the CentOS Server 5.0 is 1.27GB (1.13GB), where 1.13GB is the size of the image without the duplicate blocks. The shared size between CentOS Server 5.0 and CentOS Server 5.5 is 376MB, which does not include duplicate blocks either. Hence, to calculate what percent of CentOS Server 5.0's

Table 5.1: Virtual Disk Images from [30] and [31]

OS Name and Version	**Image Size**	**FS**
CentOS Server 5.0 i386	1.27GB (1.13GB)	ext3
CentOS Server 5.5 i386	1.32GB (1.17GB)	ext3
CentOS Server 5.8 x86_64	1.62GB (1.40GB)	ext3
CentOS Server 6.0 x86_64	0.98GB (0.77GB)	ext3
CentOS Server 6.1 x86_64	2.16GB (1.94GB)	ext3
CentOS Server 6.2 x86_64	2.18GB (1.96GB)	ext3
Debian 6.0.2.1 x86_64	0.91GB (0.76GB)	ext3
Fedora 16 x86_64	2.49GB (2.24GB)	ext3
Fedora 17 x86_64	2.66GB (2.40GB)	ext3
RHEL 6.0 x86_64	1.50GB (1.36GB)	ext4
RHEL 6.1 x86_64	1.80GB (1.66GB)	ext4
RHEL 6.2 x86_64	1.80GB (1.70GB)	ext4
Ubuntu Server 9.10 i386	0.90GB (0.75GB)	ext3
Ubuntu Server 10.04 i386	0.85GB (0.74GB)	ext3
Ubuntu Server 11.04 i386	0.92GB (0.78GB)	ext3
Ubuntu Server 11.10 i386	1.00GB (0.84GB)	ext3
Ubuntu Server 12.04 i386	1.05GB (0.85GB)	ext3
Windows Server 2008 32bit	19.0GB (6.57GB)	NTFS
Windows Server 2008 64bit	21.0GB (10.1GB)	NTFS
Windows Server 2008 R2	20.0GB (8.60GB)	NTFS
Windows Server 2008 R2 SQL	21.5GB (10.5GB)	NTFS

content appears in CentOS Server 5.5, we take $\frac{376MB}{1.13GB} = 0.33$ or 33%.

Content similarity for Red Hat Enterprise Linux VMs is between 38% and 56%; for Fedora VMs, content similarity is 30%. Content similarities are fairly high, considering that the VMs have different version numbers. Later, we will discuss how content similarity is higher for VMs that have the same OS and version numbers, but have different packages installed to perform different tasks. Content similarity among CentOS Servers 5.0 through 5.8 is also approximately 30%. Further, as shown in Table 5.2, there is no content similarity between CentOS VMs with version numbers 5.8 and earlier, and those with version numbers 6.0 and later. There is much higher content similarity between CentOS Servers 6.1 and 6.2 (1.15GB or 60%). The content similarity is much higher between VMs with close version numbers. It is not shown in Table 5.2, but the content similarity between CentOS Servers 5.5 and 5.7 is 550MB, while the content similarity between CentOS Servers 5.5 and 5.8 is 444MB. CentOS Server 6.0 is a minimal version of the server; hence, there is very low content similarity between it and the later versions of CentOS Server.

For most OSes, we note that content similarity goes down drastically after each major release. For example, that was true for Ubuntu VMs, for which major releases of Ubuntu

Table 5.2: Content Similarity between VMs

VM 1		VM 2		Shared
CentOS 5.0	33%	CentOS 5.5	32%	376MB
CentOS 5.0	28%	CentOS 5.8	23%	376MB
CentOS 5.0	0%	CentOS 6.0, 6.1, 6.2	0%	0MB
CentOS 5.5	38%	CentOS 5.8	32%	444MB
CentOS 5.5	0%	CentOS 6.0, 6.1, 6.2	0%	0MB
CentOS 5.8	0%	CentOS 6.0, 6.1, 6.2	0%	0MB
CentOS 6.0	36%	CentOS 6.1	15%	280MB
CentOS 6.0	28%	CentOS 6.2	12%	220MB
CentOS 6.1	60%	CentOS 6.2	60%	1.15GB
Fedora 16	33%	Fedora 17	30%	720MB
RHEL 6.0	56%	RHEL 6.1	46%	730MB
RHEL 6.0	50%	RHEL 6.2	38%	650MB
RHEL 6.1	56%	RHEL 6.2	53%	890MB
Ubuntu 9.10	0%	Ubuntu 10.04, 11.04, 11.10, 12.04	0%	0MB
Ubuntu 10.04	18%	Ubuntu 11.04	17%	132MB
Ubuntu 10.04	14%	Ubuntu 11.10	12%	100MB
Ubuntu 10.04	7.6%	Ubuntu 12.04	6.6%	56MB
Ubuntu 11.04	26%	Ubuntu 11.10	24%	204MB
Ubuntu 11.04	13%	Ubuntu 12.04	12%	100MB
Ubuntu 11.10	16%	Ubuntu 12.04	16%	136MB
Win 2008 32b	67%	Win 2008 64b	44%	4.4GB
Win 2008 32b	23%	Win 2008 R2	17%	1.5GB
Win 2008 32b	23%	Win 2008 R2 SQL	14%	1.5GB
Win 2008 64b	31%	Win 2008 R2	36%	3.1GB
Win 2008 64b	31%	Win 2008 R2 SQL	30%	3.1GB
Win 2008 R2	96%	Win 2008 R2 SQL	79%	8.3GB

Server were 8.04 and 10.04. Ubuntu Server 8.04 and Ubuntu Server 10.04 are Long Term Support (LTS) releases of Ubuntu. As with CentOS Server VMs, there was no content similarity between VMs with version numbers prior to 10.04, and the ones with 10.04 and later. Indeed, compared to other OSes, content similarity between different versions of Ubuntu Server VMs is very low.

For Windows servers, Windows Server 2008 R2 was released after Windows Server 2008, and is a 64-bit-only OS. Hence, the content similarity between R2 and Windows Server 2008 64-bit is higher than the content similarity between R2 and Windows Server 2008 32-bit. The Windows Server 2008 R2 SQL VM has SQL Server Express 2008 & IIS installed. The content similarity between R2 and R2 SQL is very high, even after R2 SQL has been customized with new applications.

So far in this section, we have compared base VMs. For CentOS Servers, content similarity

Table 5.3: Content Similarity between CentOS Servers with All Updates

VM Name	Image Size	Shared Size
CentOS Server 5.0	2.3GB (1.9GB)	1.0GB
CentOS Server 5.5	2.4GB (1.8GB)	
CentOS Server 6.1	3.5GB (2.9GB)	2.2GB
CentOS Server 6.2	3.7GB (2.9GB)	

increased after the latest updates were applied. Table 5.3 shows comparisons after application of updates to CentOS Servers. Content similarity between CentOS Server 5.0 and CentOS Server 5.5 increased from about 33% to 53%. For CentOS Servers 6.1 and 6.2, it went up from 60% to 75%. This gain comes at a cost, because the overall sizes of the CentOS VMs increased by about a GB after the updates were applied. Similar behavior was not observed for other OSes.

5.2 Customized VM Comparison

In this section, we compare VMs that have the same OS and version numbers, but are customized with different packages. In the previous section, we saw that content similarity between VMs with the same OSes but different versions is not always very high. But in data centers, hundreds of thousands of VMs are deployed daily, and hence there will be opportunities to schedule VMs with both the same OSes and the same versions at the same hosts. Typically, certain versions of each OS will be more widely used than other versions, and users may have their own customized VMs of that version. To compare these types of VMs, we started with the same base image, installed different sets of packages on that image to create customized VMs, and measured the content similarity between them.

Table 5.4 describes the set of packages we used to customize Ubuntu VMs. For each version of Ubuntu Server, we created three VMs from the same base image. Then, for each of the VMs, we installed different subsets of the packages shown in Table 5.4. We selected at least one package from each category in the table for each custom VM. The comparison results are shown in Table 5.5. In the last column of that table, we display the average sizes of the three customized VMs after installation of the packages, without the duplicate blocks. In the second column, we averaged the content similarities in each pair of VMs. Even after we installed very different sets of packages, the content similarity between VMs with the same OS versions remained very high.

For Fedora VMs, we followed a similar technique to customize the VMs. The results, also shown in Table 5.5, are very similar to the Ubuntu results.

Table 5.4: List of Packages Used to Customize Ubuntu Server VMs

Ubuntu packages	
Web Servers	Apache2 Web Server, Squid Proxy Server
Databases	MySQL, PostgreSQL
Wiki Apps	Moin Moin, MediaWiki
File Servers	FTP Server, CUPS Print Server
Email Services	Postfix, Exim4
Version Control System	Subversion, CVS, Bazaar

Table 5.5: Average Content Similarity between Customized Ubuntu Server VMs and Fedora VMs

OS Name & Version	**Avg. Similarity**	**Avg. Size**
Ubuntu Server 10.04	680MB	920MB
Ubuntu Server 11.10	925MB	1.15GB
Ubuntu Server 12.04	870MB	1.09GB
Fedora 16	2.45GB	3.1GB
Fedora 17	2.3GB	3GB

The highest content similarity between customized VMs with the same OS and version number appears in CentOS Server VMs. CentOS Server VMs come pre-installed with most packages. After all the updates to CentOS Server VMs have been applied, VMs with the same CentOS version numbers are almost identical in content. Because CentOS VMs have most packages already installed, even when those VMs are customized, the customizations tend to be small (e.g., changes in setting files). In other words, the customizations do not include installation of different packages on different VMs. For that reason, content similarity between customized CentOS Server VMs with the same version numbers remains very high (more than 90%) once all the updates have been applied.

5.3 Conclusions

The main lessons to take away from this chapter are the following. The content similarity between virtual disk images depends on many different factors. There doesn't seem to be any content similarity between VMs with different operating systems. For certain OSes (CentOS, Fedora, and RHEL), the content similarity is relatively high between VMs with the same OS but different version numbers, but for other OSes (Ubuntu), that is not the case. For VMs with the same OS but different version numbers, the content similarity is highest when the version numbers are closest. The content similarity among virtual disk images that have

the same operating system and the same version number remains very high, even after the virtual disk images have been customized with different packages and user data.

In the next chapter, we describe our simulations of VM deployments in data centers using content-based scheduling algorithms. In the simulations, we used our content similarity results from this chapter to calculate data transfer savings associated with the transfer of virtual disk images between racks in the data centers.

CHAPTER 6

SIMULATION OF VM DEPLOYMENTS IN DATA CENTERS

In Chapter 3 we described several content-based scheduling algorithms we designed for VM deployments in IaaS data centers. In this chapter, we evaluated effectiveness of those scheduling algorithms by running simulations of VM deployments in data centers. In our simulations, we used our results on content similarity between virtual disk images from Chapter 5. We did two different types of simulations. In the first simulation, we measured the effectiveness of the content-based scheduling algorithm based on a VM deployment trace collected from a real cloud service provider. In the second simulation, instead of using that trace, we used varying VM deployment rates, and measured their effects on network bandwidth savings and utilization of compute nodes. The purpose of the first simulation was to see how well the content-based scheduling algorithm performed in a real cloud setup. The second simulation was intended to serve as a sensitivity analysis to see how VM deployment rates and node capacities affect the overall performance of the algorithm.

6.1 Simulation of VM Deployments Based on an IaaS Data Center Trace

6.1.1 Simulation Setup

For our simulation, we generated a VM deployment trace. The trace consisted of all the VMs deployed during the simulation. Each VM deployment event in the trace contained the following information: start and termination time, OS name and version number, and instance type. Next, we describe how we assigned each of those properties to the VMs.

VM deployment rates

We wanted to use realistic estimates of how many VMs were getting deployed at any given time of day. We followed the technique described in [2] to estimate the VM deployment

Table 6.1: Description of VM Instance Types

Instance type	**Resource Usage**
Small	1.70GB RAM, 1 Compute Unit
Medium	3.75GB RAM, 2 Compute Units
Large	7.50GB RAM, 4 Compute Units
XLarge	15.0GB RAM, 8 Compute Units

rates. According to [2], for AWS data centers, given two AMI IDs and their start times, it is possible to calculate the number of VMs that were deployed in the same AWS data center between the start times of the two VMs. Following that technique, we periodically deployed new VM in the AWS Virginia data center every five minutes for 24 hours. From the resulting data, we were able to gather the number of VMs deployed in each 5-minute period in one day. We used those data in generating start times and the number of VMs to deploy in our simulation.

Although we were able to estimate the start times of VMs, we could not perform any similar experiment to estimate the duration or termination times of the VMs in commercial clouds. Using all the information we had available, we estimated that a small portion of the VMs last only a few hours, and another small portion of the VMs stay in operation for weeks to months. Hence, in our simulation, we set the duration for 25% of the VMs to be between a few minutes to a couple of hours; specifically, we uniformly selected time lengths between 5 minutes and 2 hours for those VMs. We also set 15% of the VMs to have a duration between one day and a few weeks. Since we simulated one week of VM deployments in a data center, any durations longer than one week had the same effect. For the rest of the VMs, meaning the majority of the VMs, we uniformly selected running times between two hours and 15 hours.

OS distribution

Since many websites are hosted in cloud data centers, we used the distribution of Web server OSes listed in [32] to assign OSes to VMs in our simulation. However, [32] does not provide a breakdown of distributions by version numbers. Therefore, we assigned much higher probabilities to the latest versions of the OSes in our simulation trace than to the older versions (e.g., 70% for CentOS 6.2, 75% for Fedora 17, and 50% for Ubuntu 12.04).

VM instance types

VM instance types specify the resource allocation for VMs when they are deployed. Table 6.1 shows different instance types. Like the duration times of VMs, information about VM instance type distributions in data centers is not publicly available. Hence, we assigned instance types to VMs in our simulation based on the numbers that were provided in [33].

Data center

In our simulation, we simulated deployment of VMs in a single data center. We implemented a data center architecture that was inspired by a real cloud architecture deployed by a major U.S. ISP and is shown in Figure 6.1. Each node was represented by a blade server, and each rack contained several blade servers. In our deployment, the amount of RAM was the limiting resource of the blade servers; therefore, the RAM determined the number of VMs that could be hosted on a single blade server. The blade servers in our data center were equipped with 140GB of RAM. Table 6.1 shows the resource requirements for each VM instance type. As shown in the table, we decided to use EC2 Compute Units as the measure of CPU requirements. Given the blade server specs and Table 6.1, we could deploy at most 9 XLarge VMs simultaneously on a single blade (compute node). Alternatively, we could have a combination of at most 2 XLarge, 6 Large, and 17 Medium VMs deployed at the same time ($2 \cdot 15 + 6 \cdot 7.5 + 17 \cdot 3.75 = 138.75 \leq 140$). During the simulation, at a VM's termination time, we removed the VM from the node, and the resources occupied by the VM became available again.

Content similarity

During the simulation, scheduling algorithm decisions were based on content similarity between the VM being scheduled and the rest of the VMs running on the compute nodes. We calculated the content similarity between VMs as follows. When there was a request from the scheduling algorithm to calculate content similarity between VM_1 and VM_2, we looked at two things. First, if the two VMs had different OSes, we returned zero content similarity. If the two VMs had the same OS but different versions, we returned the size of the shared content noted in Table 5.2. If the request was for two VMs with the same OS and the same version, then we treated the two VMs as customized VMs based on the same base image. In Chapter 5, we show content similarity mostly between customized VMs for Ubuntu and Fedora VMs, but we have collected similar results for VMs with other OSes as well. For customized VMs of each OS and version number, we have four different content

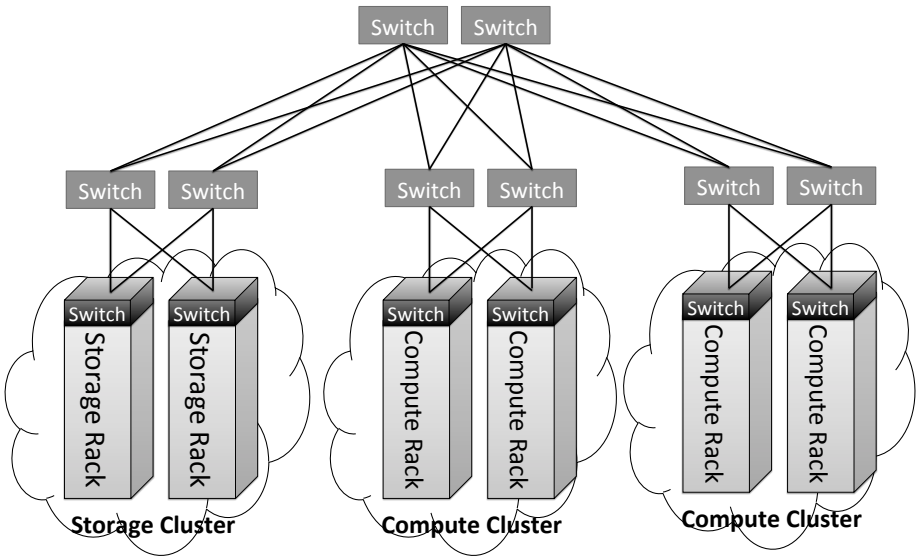

Figure 6.1: Data center architecture

similarity values for each pair of comparisons. In response to a content similarity request for a pair of customized VMs, we returned a number uniformly chosen between the minimum and maximum of the four comparison values.

6.1.2 Simulation Results

We ran the simulation with different scheduling algorithms. The greedy and dedicated node algorithms are described in Section 3.3. We tried the dedicated node algorithm with $n = 1, 5$, and 10, where n is the number of nodes we evaluated to find the greatest content similarity. We also used a random algorithm, which worked as follows. A random node in the data center was selected, and a local VM from that node that had the highest content similarity to the VM being scheduled was used to determine which other blocks needed to be transferred from the new VM to that selected node.

In the simulation, the amount of data transferred was calculated as follows. Let VM_{new} be the VM disk image we were transferring to a node. The VM on that node with the highest content similarity to VM_{new} is VM_{local}. Let $size_{orig}$ be the size of the VM_{new} on the file system; $size_{dist}$ be the size of the VM_{new} without the duplicate blocks; and $size_{shared}$ be the size of shared content between VM_{new} and VM_{local}. To transfer VM_{new}, it is only necessary to transfer missing blocks to the destination, and each unique disk block is transferred only once. Hence, the amount of data transferred to the destination is $size_{dist} - size_{shared}$.

The simulation results are shown in Figure 6.2. The simulations consisted of deploying

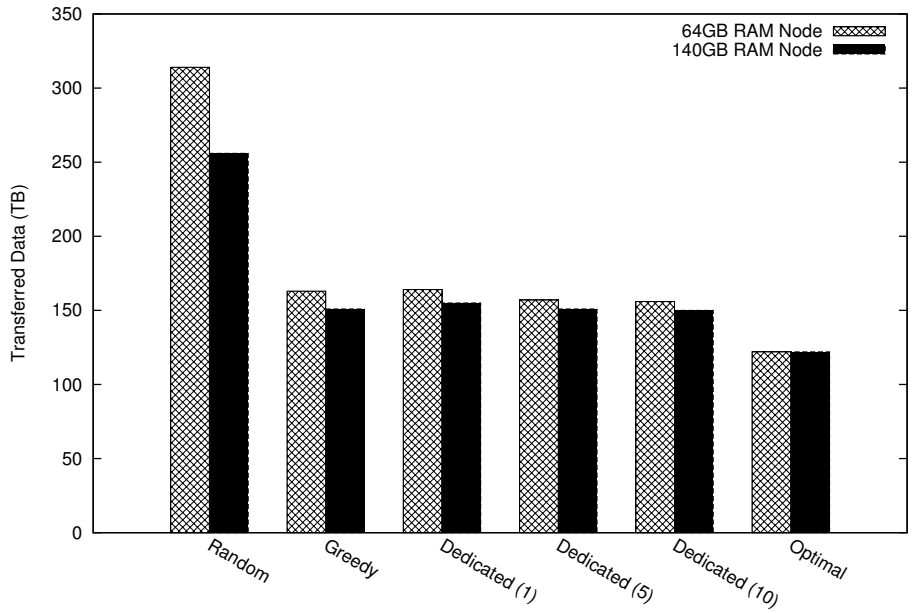

Figure 6.2: Total amount of data transferred for different scheduling algorithms. The total size of all VMs deployed is 578TB, and the size of all VMs with duplicate disk blocks eliminated is 455TB. The confidence interval bound in either direction is less than 1TB for algorithms involving randomness (Dedicated, Random)

VMs for a duration of one week. Initially, we started with a data center with no VMs running. We ran simulations with two data center setups, one with 140GB RAM nodes, and another with 64GB RAM nodes. We used the 64GB nodes to see how node RAM capacities affected the different algorithms. Each run of the simulation was performed on the same trace file that we generated.

The total size of all the deployed VMs was 578TB, and the size without the duplicate blocks was 455TB. Figure 6.2 shows the total amount of data transferred while the virtual disk images was being copied during the VM deployments. In the bars labeled *Optimal*, we show how much data would have to be transferred if, during the deployment of each VM, there existed another VM with the highest possible content similarity, and the new VM was scheduled on the same node as the other VM. Such a scenario cannot be guaranteed in a real scheduling algorithm execution, because there is no guarantee that there will be another VM with the highest content similarity running when each VM is deployed or that there is space on the compute node with this VM.

It can be seen in Figure 6.2 that the scheduling algorithms performed better when 140GB RAM nodes were used. The difference is the largest for the random algorithm. That was expected, because 140GB RAM nodes can host more than twice as many VMs as the 64GB

RAM nodes can. In the random algorithm, VMs are scheduled to random nodes. Since a 140GB RAM node can host more VMs than a 64GB RAM node can, a randomly chosen node is more likely to have a VM with higher content similarity. Other algorithms are not affected as much, because the scheduler in the other algorithms evaluates multiple nodes to find the one with the highest content similarity.

From here on, we will refer to the results from the simulation with the 140GB RAM nodes. The random algorithm transferred 256TB, where the total size of the VMs in the file system was 578TB. Even the random algorithm decreases the amount of transferred data by $1 - \frac{256TB}{578TB} = 56\%$. The dedicated node algorithms with $N = 1, 5$, and 10 decreased the amount of data transferred by $1 - \frac{155TB}{578TB} = 73.2\%$, $1 - \frac{151TB}{578TB} = 73.9\%$, and $1 - \frac{150TB}{578TB} = 74\%$, respectively. The greedy algorithm decreased the amount by $1 - \frac{151TB}{578TB} = 73.9\%$.

The greedy algorithm performed only as well as the dedicated nodes algorithm with $N = 5$. The greedy algorithm evaluates all the nodes in the data center, while the dedicated nodes algorithm only evaluates 5 nodes with the same OS as the VM being deployed. Hence, in terms of how long it takes the scheduling algorithm to find the destination node, the dedicated node algorithms have a big advantage over the greedy algorithm. But, as we mentioned earlier, the greedy algorithm does not have the same restriction as the dedicated node algorithm, where each node can only run VMs with the same OS.

There was a small gain in going from $N = 1$ to $N = 5$ in the dedicated node algorithms, and an even smaller gain in going from $N = 5$ to $N = 10$. That tells us that we do not need to evaluate many dedicated nodes to find VMs with high content similarity.

The results are very promising. Transferring 73% less network data between racks inside a data center is a big improvement. The cost is that there is extra computation involved in transferring the VMs to the destination nodes. If the network bandwidth is the bottleneck inside a data center, then the benefits of the content-based scheduling algorithm are significant.

6.2 Analyzing Affects of Deployment Rates

In this section, we describe a simulation in which we measured performance of the scheduling algorithm in terms of how much data is saved transferring VMs to destination hosts when they are deployed, and the average utilization of host nodes, as functions of the deployment rates of VMs and the capacity of physical nodes in the data center. The main goal of the simulation was to do a sensitivity analysis to see how deployment rates and node capacities affect the overall performance of our scheduling algorithm. In this section, except for the

deployment rates, simulation parameters are same as in Section 6.1.1.

6.2.1 Simulation Results

The simulation measured two things. First, it determined what percent of the total amount of data was transferred during the deployment of VMs that used content-based scheduling algorithm. Specifically, it looked at the amount of data transferred during copying of VM disk images from storage nodes to compute nodes compared to total size of all VM disk images. For example, if a particular VM's size is 2GB, but because of high content similarity, only 750MB was transferred during deployment of that VM, then 37.5% of the data was transferred.

Second, the simulation measured the average utilization of all the blade servers in the data center over the period of time the simulation ran. To measure utilization, we divided the number of VMs that were actively deployed on a particular blade server by the maximum number of VMs that can run on the blade server. Blades that are idle (do not have any VMs running) did not contribute to the overall average utilization calculation.

We started the simulation with all blade servers idle. The simulation captured VM deployments in a single data center during a five-day window.

Figure 6.3 shows average percentages of data transferred during deployment of VMs in the simulation for the content-based algorithm and random algorithm with various parameters. For each of the simulations, there were 10 repetitions. A confidence interval (CI) for each simulation is displayed in the plot. CIs are difficult to see in the plot, because the intervals are very close.

Parameters that were varied in the simulations are as follows. `c:40` and `c:80` represent the varying capacities of physical blades in the data center. `c:40` means that one blade server can simultaneously host 40 small (or 20 medium, or 10 large, or 5 xlarge) VMs. `c:80` means that a blade server has twice the capacity of a `c:40` blade server. These labels correspond to blade servers with 64GB and 140GB of RAM (discussed in Section 6.1.1).

n:1 and `n:5` are varying parameters in the *Dedicated Node* scheduling algorithm. `n` is the number of blade servers that are selected at random in the *Dedicated Node* algorithm. In the case of `n:1`, one blade server is selected at random; then, we choose the VM from that one server that has the highest content similarity to the VM being deployed, and use it to transfer the new VM disk image difference to that blade server.

Parameters `a:1.01` and `a:1.2526` are deployment rates of VMs in the data center. We wanted to study how the performance of the scheduling algorithm would be affected, if, at each VM deployment time, multiple VMs were deployed simultaneously. To determine the

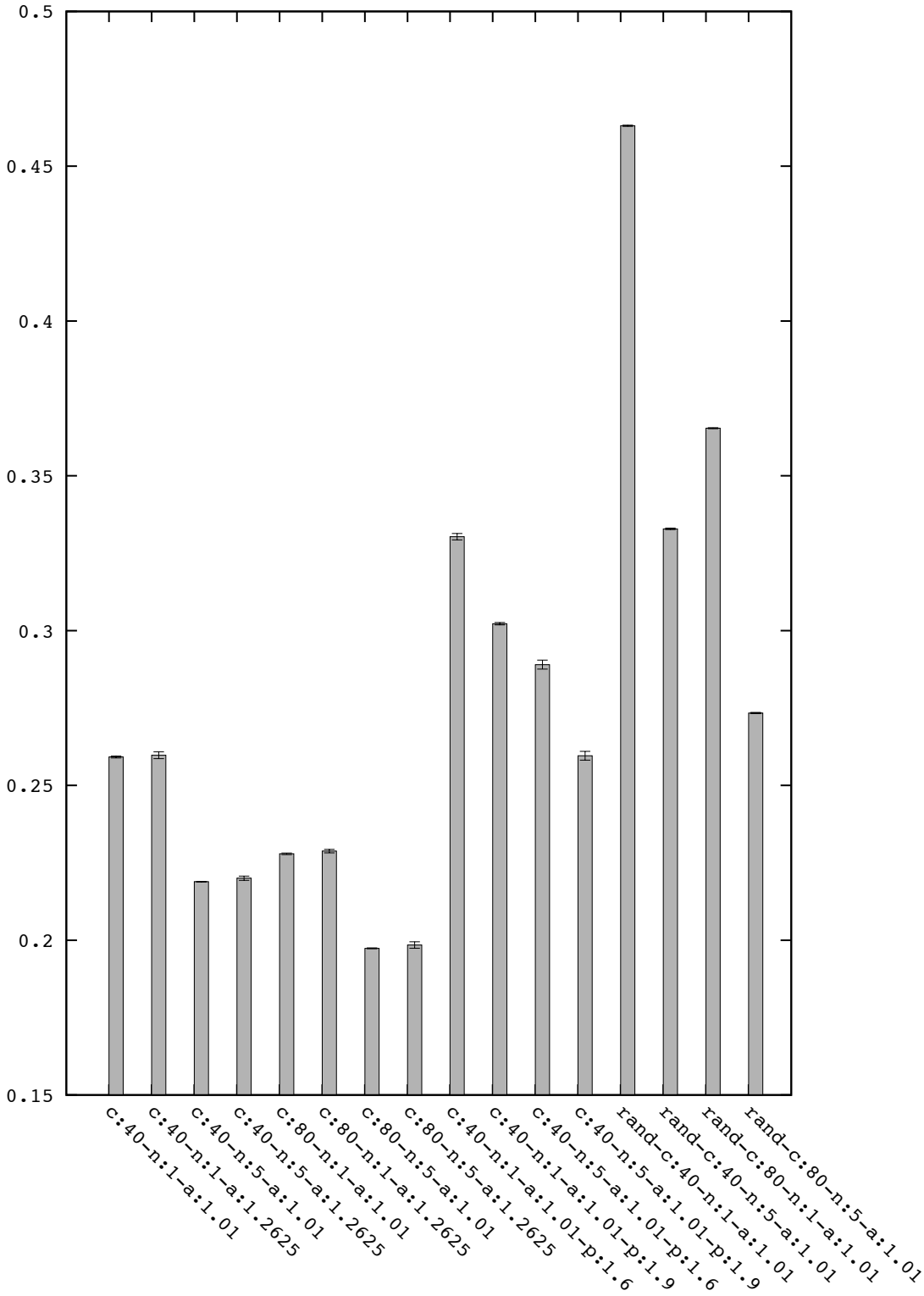

Figure 6.3: Average percent of data transferred. `c` is the blade server capacity; `n` is the number of blade servers selected in the scheduling algorithm; `a` is the deployment rate of VMs; `p` is the Pareto rate for bulk deployment; and `rand` is a random algorithm

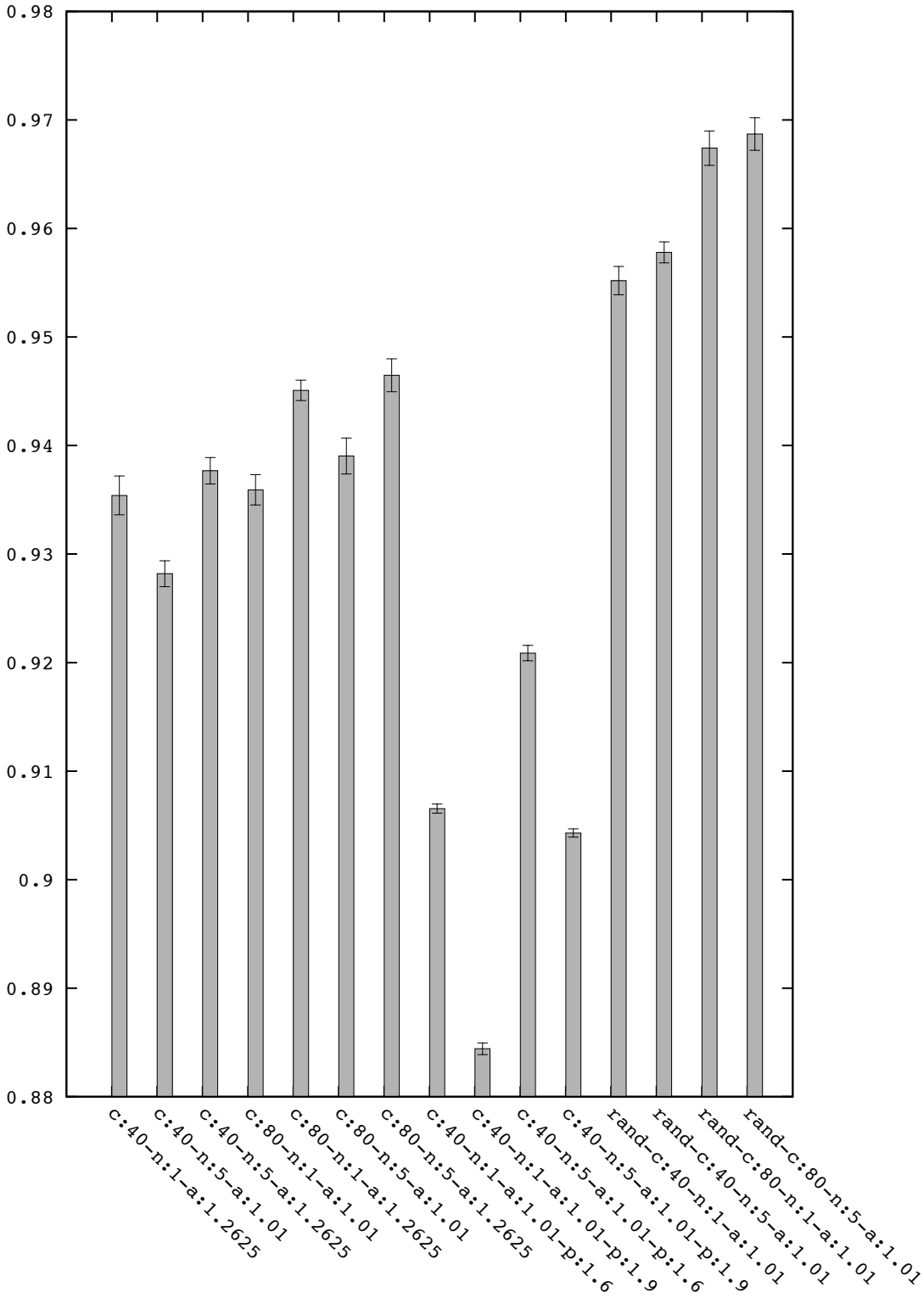

Figure 6.4: Average utilization of server blades. `c` is the blade server capacity; `n` is the number of blade servers selected in the scheduling algorithm; `a` is the deployment rate of VMs; `p` is the Pareto rate for bulk deployment; and `rand` is a random algorithm

Table 6.2: Sign Table: Effects of `c`, `n`, and `a` on Average Percent of Data Transferred

	Effects							
Simulation	c	n	a	cn	ca	na	cna	% Data Sent
c:40-n:1-a:1.01	−	−	−	+	+	+	−	0.259
c:40-n:1-a:1.2625	−	−	+	+	−	−	+	0.260
c:40-n:5-a:1.01	−	+	−	−	+	−	+	0.219
c:40-n:5-a:1.2625	−	+	+	−	−	+	−	0.220
c:80-n:1-a:1.01	+	−	−	−	−	+	+	0.228
c:80-n:1-a:1.2625	+	−	+	−	+	−	−	0.229
c:80-n:5-a:1.01	+	+	−	+	−	−	−	0.197
c:80-n:5-a:1.2625	+	+	+	+	+	+	+	0.198

Table 6.3: Effect Estimate: % Data Transferred

Effect	Effect Size
c	-0.10536
n	-0.14090
a	0.00366
cn	0.01914
ca	0.00022
na	0.00076
cna	-0.00036

Table 6.4: Effect Estimate: Average Utilization

Effect	Effect Size
c	0.0437
n	0.0133
a	0.0399
cn	-0.0043
ca	-0.0069
na	-0.0061
cna	0.0027

number of VMs to deploy at each deployment time, we chose a Pareto distribution; usually, only a small number of VMs are deployed at once, but on rare occasions, some organizations may request many VMs at the same time. `p:1.6` and `p:1.9` represent the rate for this Pareto distribution.

In Figure 6.3, in the last four bars, `rand` stands for the random algorithm. In the random algorithm, once we have selected a random host node, we must still find the local VM with the highest content similarity to the VM being deployed. However, the blades are not dedicated to run any specific type of OS; i.e., VMs with different OSes can run on the same blade.

In Figure 6.4, for the same simulations described above, average utilization of the blade servers is shown.

In Figure 6.3, we can see that in our scheduling algorithm, on average, we have to transfer only about 20% to 27% of the total data without the Pareto distribution, and 27% to 33% with the Pareto distribution. That means that content-based scheduling can decrease the network bandwidth associated with deployment of VMs in the data center by more than 70% in most cases.

A random algorithm can perform quite well in some cases, e.g., when `c:80` (140GB RAM)

blade servers are used and 5 blade servers are selected at random. That would be expected, because when we select 5 blade servers with higher capacities, it increases the probability that a local VM in one of those blades will have high content similarity to the VM being deployed.

The figure shows that in all scenarios, higher blade capacity usually results in much less data being sent over the network.

Figure 6.4 shows that the random algorithm performs the best, as expected, when it comes to average utilization. The content-based scheduling algorithm with the Pareto distribution for multiple simultaneously deployed VMs performs the worst. The reason is that when we have to deploy multiple VMs at the same time, we try to co-locate all of the VMs at the same blade server. In most cases, that results in all of the VMs being deployed on an empty blade server; hence, the average utilization goes down.

Table 6.2 shows the effects of parameters `c`, `n`, `a`, and combinations of them on performance of the content-based scheduling algorithm. The estimated effects on percent of data transferred are shown in Table 6.3. Table 6.4 shows the estimated effects on average utilization. These estimated effects are based on the simulation results shown in Table 6.2. Our purpose in developing the estimates was to study how the three parameters affect the performance of our content-based scheduling algorithm in terms of data transfer savings and average utilization of the blade servers.

Table 6.3 shows that the capacity of the blade servers and the parameter `n` used in the scheduling algorithm have the biggest impact on network bandwidth savings associated with deployment of VMs. It means that instead of having many inexpensive blade servers with lower capacities, having fewer but higher-capacity blade servers will yield higher network bandwidth savings. A data center designer would have to look at the trade-offs between using cheaper blade servers with more expensive network switches, or more expensive blade servers with cheaper network switches.

Table 6.4 shows that the capacity of the blade servers and the deployment rate of the VMs have the biggest impact on average utilization of blade servers. It is not surprising that higher deployment rates lead to higher average utilization. However, having higher-capacity blade servers also leads to higher average utilization.

6.3 Conclusions

There are several important lessons to draw from the simulation results in this chapter. One of the biggest is that content-based scheduling can lead to significant decreases in net-

work traffic associated with VM transfers between racks. Of course, it comes at the cost of extra computation. However, even without employing computation-intensive scheduling algorithms, we can still achieve high network bandwidth savings using content-based scheduling of VMs in data centers. The dedicated node algorithm with $N = 1$ performed almost as well as the one with $N = 5$. That tells us that if we let each compute node be dedicated to host VMs with the same OS (but not necessarily the same OS release version), then the scheduling algorithm can perform very well by just randomly selecting a compute node whose OS matches the VMs being scheduled. In other words, the algorithm doesn't have to examine more than one compute node to find high content similarity between VMs being scheduled and VMs already running on dedicated compute nodes.

The results from Chapters 5 and 6 also suggest that content similarity between VMs can sometimes be found using techniques even simpler than Bloom filters. For example, matching of the OS name and release versions on VMs can also lead to some content similarity between the VMs. The reason is that, as we saw in Chapter 5, most content similarity seems to exist between custom VMs with the same origin OS and version.

The above findings show that content-based scheduling can easily be plugged into existing cloud data centers, and can lead to noticeable network bandwidth savings.

CHAPTER 7

CONCLUSION

Cloud computing makes it easy to deploy and terminate virtual machines as desired, and the pay-for-what-you-use billing model encourages users to keep VMs running only when needed. Hundreds of thousands of VMs may be deployed in a day in a large cloud data center. Because of large sizes of virtual disk images, intra-data-center traffic due to VM deployments can put a significant strain on a data center's network infrastructure. In this thesis, we presented a novel scheduling algorithm that utilizes similarity between virtual disk images, a similarity that is maintained for VMs with the same OS and version number even if the VMs are customized and are in use. We quantified the similarities between VM images and showed that the similarity can be as high as 60–70%, or even over 90% in some cases. We used simulations to demonstrate that our scheduling algorithm can reduce the network utilization associated with the virtual disk image transfers by over 70%. Such savings are significant enough to affect the networking design for cloud data centers, and definitely can reduce network congestion and increase the available bandwidth for the VMs running in the cloud data center. Since the optimization results in co-location of VMs with shared pages on the same compute node, it also increases the benefits of using memory page sharing on the node, resulting in better utilization of the often-bottlenecked memory resources.

Alongside the rise of cloud computing, we have seen exponential growth in smart phone and tablet ownership. As individuals become more and more reliant on their mobile devices to perform day-to-day tasks, it is important to back up content on those devices regularly to protect against device loss or theft. At the same time, backing up content makes it easier to migrate from one device to another (e.g., upgrade to a new smart phone or tablet). We have presented a context-aware personal storage cloud that takes advantage of all of a user's devices (mobile and PC) as well as online cloud storage providers when necessary to provide a seamless and cost-efficient personal storage and backup solution. We rely on context information (network, storage nodes, file usage) to optimize the file placement decisions (when and where). We employed techniques similar to those used by our content-based scheduling algorithm for VMs to take advantage of content similarity among a user's devices to minimize the cost of personal storage.

Today, the way we do computing is changing rapidly. For that reason, we believe that what we have presented in this thesis can have a big impact in both cloud computing and mobile computing. We demonstrated how our content-based scheduling algorithm can significantly reduce the network traffic in data centers, and how we can make a personal storage cloud for mobile devices cost-effective through storage and network optimizations. We also showed how the same approach can be applied to both areas to achieve such optimizations. Further studies of those techniques, especially content similarity analysis, can lead to further optimizations in other areas.

REFERENCES

[1] H. Liu, "Amazon data center size," published March 13, 2012. [Online]. Available: http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/

[2] T. von Eicken, "Amazon usage estimates," published October 5, 2009. [Online]. Available: http://blog.rightscale.com/2009/10/05/amazon-usage-estimates/

[3] A. Mitchell, T. Rosenstiel, L. H. Santhanam, and L. Christian, "The explosion in mobile audiences and a close look at what it means for news," published October 1, 2012. [Online]. Available: http://www.journalism.org/analysis_report/future_mobile_news

[4] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534540 pp. 7:1–7:12.

[5] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine co-migration for the cloud," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996153 pp. 159–170.

[6] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996151 pp. 135–146.

[7] K. Takahashi, K. Sasada, and T. Hirofuchi, "A fast virtual machine storage migration technique using data deduplication," in *Proceedings of CLOUD COMPUTING 2012: The 3rd Int. Conf. on Cloud Computing, GRIDs, and Virtualization*, 2012, pp. 57–64.

[8] S. K. Bose, S. Brock, R. Skeoch, and S. Rao, "CloudSpider: Combining replication with scheduling for optimizing live migration of virtual machines across wide area networks," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2011, pp. 13–22.

[9] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers," in *Proceedings of IEEE INFOCOM, 2012*, 2012, pp. 181–189.

[10] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1508293.1508299 pp. 31–40.

[11] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," in *Proceedings of the Linux Symposium*, 2009, pp. 19–28.

[12] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1831407.1831429

[13] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman, "Satori: Enlightened page sharing," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*. USENIX Association, 2009.

[14] Apple Inc., "Storage Upgrade Options," visited March 5, 2013. [Online]. Available: http://www.apple.com/icloud/includes/lightbox-storage.html

[15] Google Inc., "My Backup Pro," visited March 15, 2013. [Online]. Available: https://play.google.com/store/apps/details?id=com.rerware.android.MyBackupPro

[16] BullGuard, "BullGuard Mobile Backup," visited March 15, 2013. [Online]. Available: http://www.bullguard.com/products/bullguard-mobile-backup-12.aspx

[17] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, vol. 4, 2002.

[18] E. Sit, J. Cates, and R. Cox, "A DHT-based backup system," in *Proceedings of the 1st IRIS Student Workshop*, vol. 131, 2003, p. 146.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[20] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.

[21] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 285–298, 2002.

[22] B. T. Loo, A. LaMarca, and G. Borriello, "Peer-to-peer backup for personal area networks," *Departmental Papers (CIS)*, p. 334, 2003.

[23] E. B. Nightingale and J. Flinn, "Energy-efficiency and storage flexibility in the Blue File System," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 363–378.

[24] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the Coda File System," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 3–25, Feb. 1992. [Online]. Available: http://doi.acm.org/10.1145/146941.146942

[25] "OpenStack," visited on November 10, 2012. [Online]. Available: http://www.openstack.org/

[26] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, July 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[27] "Bloom filter," Wikipedia, modified March 29, 2013. [Online]. Available: http://en.wikipedia.org/wiki/Bloom_filter

[28] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," in *Internet Mathematics*, vol. 1, no. 4. Taylor & Francis, 2004, pp. 485–509.

[29] S. Shankland, "Google: 500 million Android devices activated," CNET, published Septermber 12, 2012. [Online]. Available: http://news.cnet.com/8301-1035_3-57510994-94/google-500-million-android-devices-activated/

[30] "VMware images," Thoughtpolice, visited on November 12, 2012. [Online]. Available: http://www.thoughtpolice.co.uk/vmware/

[31] "Amazon Web Services," visited March 17, 2013. [Online]. Available: https://aws.amazon.com/

[32] "Usage of operating systems for websites," W3Techs, visited on November 13, 2012. [Online]. Available: http://w3techs.com/technologies/overview/operating_system/all

[33] T. von Eicken, "More servers, bigger servers, longer servers, and 10x of that," published on August 4, 2010. [Online]. Available: http://blog.rightscale.com/2010/08/04/more-bigger-longer-servers-10x/