Large Installation Administration Project

PERFORMANCE OF REMOTE STORAGE FOR CLOUDS

Authors:
Danny Groenewegen
danny.groenewegen@os3.nl
Pieter Lange
pieter.lange@os3.nl

Abstract

The physical location of data is sometimes a problem for companies that want to use cloud services. We have developed a way to bootstrap systems in the cloud for remote storage using iSCSI and a IPsec tunnel and have tested the performance of remote storage systems using different performance characteristics (estimations of public and private clouds). Network latency definitely has an impact on performance no matter what kind of filesystem is used, but when dedicated and reliable connections are available there are still use cases for this.

March 28, 2011
CONTENTS

5 Conclusion 14
  5.1 Future work ................................................. 15

A IPsec configuration files 16
  A.1 /etc/ipsec-tools.conf ....................................... 16
  A.2 /etc/racoon/racoon.conf ................................... 16

B Test script 17

C Reference metrics test script 19
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Baseline read speeds</td>
<td>8</td>
</tr>
<tr>
<td>(a) ext3</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>(b) ext4</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>(c) xfs</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>(d) btrfs</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>3.2</td>
<td>Baseline write speeds</td>
<td>9</td>
</tr>
<tr>
<td>(a) ext3</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>(b) ext4</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>(c) xfs</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>(d) btrfs</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Public cloud write speeds</td>
<td>12</td>
</tr>
<tr>
<td>(a) ext3</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>(b) ext4</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>(c) xfs</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>(d) btrfs</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>4.2</td>
<td>Private cloud write speeds</td>
<td>13</td>
</tr>
<tr>
<td>(a) ext3</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>(b) ext4</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>(c) xfs</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>(d) btrfs</td>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Baseline Bonnie++ results ....................................................... 7
3.2 Private cloud round-trip time and bandwidth ............................. 10
3.3 Public cloud round-trip time and bandwidth .............................. 10
4.1 Public cloud Bonnie++ results .................................................. 11
4.2 Private cloud Bonnie++ results .................................................. 13
Chapter 1

Introduction

Cloud computing infrastructures are becoming an increasingly important tool for organisations to save on expenses. Because of the virtualised nature of the resources provided by cloud systems these resources can be consolidated and equipment is more efficiently used.

However, tradeoffs have to be made before moving your organisation to the cloud. By deploying your systems in the cloud instead of maintaining your own infrastructure you relinquish some amount of control over your systems. Processing power, memory space and I/O bandwidth has to be shared between your own organisation and others. As long as the cloud provider is not over-consolidating resources, this often does not pose a problem. The fact that all persistent storage is under control of the cloud provider does make some organisations worry about where the data is physically located and if they still have any control over their data.

If organisations would provide the persistent storage themselves, they would not necessarily lose control over their data. Some organisations have access to fast (high bandwidth, low latency) links which could be used to serve storage to the cloud provider. For instance universities with access to a National Educational Network (NEN) would be likely candidates for community clouds. The Dutch SURFnet network is a prime example of such a NEN. Also, commercial organisations would like to leverage public cloud providers for their processing power and cheap scalability, but do not feel comfortable with handing over control of their data.

Research question

This made us arrive at the following research question:

*How feasible is it to securely deploy systems in cloud architectures using remote iSCSI storage?*

This question has been devided in sub-questions:

1. How do we build a secure link to the storage system in the bootstrap phase of the cloud machine?
2. What filesystems are more suitable when used over iSCSI?
3. What difficulties are involved with storage over the internet?
4. What are the security considerations involved with this solution?

We chose to limit our research to iSCSI block storage due to its availability in the marketplace and the open standards it encompasses. One of iSCSI's 'selling features' is the ability to transport it over regular TCP/IP networks. We are aware that other protocols such as NFSv4[6] are designed to handle high latency, however these don’t offer the block semantics we’re interested in.
Chapter 2

Test environment

2.1 The storage server

Our central storage system consists of a spare server in the OS3 server lab. This machine will serve generic Internet Small Computer System Interface (iSCSI) volumes over IPsec encapsulated tunnels to virtual machines booting in the cloud. This machine is supposed to be as generic as possible so we do not lock ourselves into specific use-cases. All measurements taken must be correlated to baseline tests conducted on this machine, because storage servers are usually a lot faster.

The hardware specifications of this machine are as follows:

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Pentium(R) D CPU 3.00GHz</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>2048 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Broadcom NetXtreme BCM5721 Gigabit Ethernet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk model</td>
<td>Seagate ST3808110AS</td>
</tr>
<tr>
<td>Size</td>
<td>80 GB</td>
</tr>
<tr>
<td>Cache</td>
<td>8 MB</td>
</tr>
<tr>
<td>Controller</td>
<td>Intel N10/ICH7 Family</td>
</tr>
</tbody>
</table>

2.1.1 Layers

Client data has to travel through a substantial amount of layers; each of these layers might implement buffers that we have to take into account when performing tests.

We use a dedicated disk as the iSCSI backing store with the Linux Logical Volume Manager (LVM) directly on top of the disk to flexibly partition the physical storage device into logical volumes. We chose the iSCSI Enterprise Target daemon (iETD) as the target software to serve the volumes. The native Linux IPsec implementation is serving the IPsec tunnels.

Thus, we can identify the following buffers at the storage server:

- Disk cache (physical)
- Various kernel caches, including:
– Block cache
– Page cache
– Network stack

• iSCSI Target software

2.2 The bootstrap process for clients

Because cloud providers do not offer the ability to boot from remote storage and we wanted to create a system that would be as flexible as possible (would ‘plug in everywhere’). We opted to go for a ‘local’ cloud storage disk with a minimal amount of space reserved for the kernel and an initial ramdisk. This gave us the added benefit of being able to use remaining space for a local swap disk; for obvious reasons it is undesirable to be swapping memory over the internet. This swap space can be initialised with a random cryptographic key on boot.

2.2.1 Initial ramdisk

The initial ramdisk is a temporary filesystem used in virtually every linux machine to make preparations for mounting the real root filesystem. It is tasked with loading the necessary drivers for hardware and other modules needed, configuring them and finally switching to the real root as the last step which is called pivot.root.

Our system used a generic Ubuntu 8.04 initial ramdisk, extended with IPsec and iSCSI modules. These where statically configured with a ‘hardcoded’ key for IPsec and iSCSI target to initialize; each cloud system would require its own initial ramdisk file.

2.2.2 IPsec tunnel

Using remote storage for cloud systems means that the data has to travel over unknown networks. It should be obvious that we need to ensure the origin, integrity and confidentiality of the data when it is travelling across unknown networks such as the internet. The Internet Protocol Security (IPsec) is a protocol suite for securing IP communications. IPsec provides a set of protocols to ensure this: Authentication Headers (AH) to ensure connectionless integrity and data origin authentication for IP datagrams; Encapsulating Security Payloads (ESP) to ensure confidentiality, data origin authentication and connectionless integrity; Security associations (SA) provide the bundle of algorithms and data necessary to facilitate the AH and ESP operations.

As of version 2.5.47, native IPsec support is implemented in the Linux kernel. Besides the IPsec support from the Linux kernel we used IPsec-Tools[3] to setup the connections. IPsec-Tools implements the Internet Security Association and Key Management Protocol (ISAKMP). It contains setkey, a tool to manipulate the kernel Security Policy Database (SPD) and Security Association Database (SAD). We used racoon[3] as the Internet Key Exchange (IKE) implementation. The racoon daemon runs on both the storage server and the cloud instance to negotiate a Security Association.

IPsec configuration

IPsec has two modes of operation, transport mode and tunnel mode. Transport mode is what we need for host-to-host communication, from the storage server to the cloud system. Because some of our test instances will be running behind a NAT router, AH cannot be enabled on the IPsec connection. A copy of the configuration files can be found in Appendix A.
Ipsec-Tools’ `setkey` adds the required security policies to the kernels SPD. This forces all iSCSI traffic to the storage server to be encrypted using the ESP protocol. When an encrypted connection is required, the kernel will call `raccoon` to negotiate a SA with the storage server. On both sides, `raccoon` is configured to use aes for the encryption algorithm.

### 2.2.3 iSCSI initialisation

The initial ramdisk contains the `iscsistart` binary to connect to the iSCSI target on the storage server. When connected, it tries to attach the disk to `/dev/sda` and succeeds. However, when the partition table is initialized the kernel encounters the already existing `/dev/sda1` block device initialized by Xen. This results in a kernel panic. On our local cloud we were able to create a workaround by attaching the local boot disk to `/dev/xvda`. Unfortunately this was not possible in the public cloud environment due to lack of control of the hypervisor. To enable access to the partitions of the iSCSI disk we had to map the block devices to virtual `device-mapper` devices. The `kpartx` tool allowed us to easily do this. After this alternative workaround, we were able to scan for LVM backing devices.

### 2.3 The cloud hosts

We considered two different cloud types in our test. The main difference is the location, resulting in different latencies and bandwidth.

#### 2.3.1 Private cloud

Our private cloud server is located in the same subnetwork as the storage server. It is set up with a default Xen 4.0 installation on top of Debian Squeeze. The DomU instance running on top of it is configured with 613MB of memory and 1 Virtual CPU with the same specifications as the storage server.

#### 2.3.2 Public cloud

We used Amazon EC2[2] as the public cloud provider. The EC2 instances are located in the region “EU West (Ireland)”. The instances are clones from AMI ID ami-3d1f2b49, with the manifest description “099720109477/ebs/ubuntu-images/ubuntu-lucid-10.04-amd64-server-20110201.1”. We configured the EC2 instances with the type `micro instance`. Those have 613MB of memory, 1 CPU core and up to 2 EC2 Compute Units (ECUs).
Chapter 3

Reference metrics

The performance measured using different filesystems and environments won’t have any value without a reference to compare them to. For the reference metrics we measured the performance from the storage server itself. We executed the tests directly on the LVM block devices, without iSCSI and IPsec. The results are presented in Section 3.3.

The test script used from the private and public cloud can be found in Appendix B. Besides the performance tests, we also measured the bandwidth using `iperf` between each filesystem test. During the whole test we also measured the round-trip time using ping. The round trip time and bandwidth can be found in Section 3.4.

3.1 Tools

Bonnie++

`bonnie++` is the go-to tool if you require a quick indication of a filesystems’ performance. It performs three basic throughput tests using different IO interfaces and also has the ability to test the performance in creating, reading and deleting inodes (‘files’). We only used the last feature of bonnie because Iozone gave us much more fine grained insight in the filesystem throughput performance.

Iozone

Iozone is a standard UNIX filesystem benchmarking tool that we used as our main result.

iperf

We used `iperf` to measure the network throughput. Because the filesystem performance cannot exceed this speed by definition (except for short durations due to caching), this gives us a decent expected value and enables us to see where caching must occur.

ping

High latencies might influence the filesystem performance aswell; therefore we also measure the machines’ response time throughout the test.
3.2 Buffers

Buffers are useful for queueing writes so they can be flushed to the backing store in order and in a large batch, which improves performance and consistency. However, for the purpose of testing filesystems buffers also prevent us from measuring the actual throughput of the medium when only writing very small files. These kinds of writes are the ones that will be most impacted by filesystems that use aggressive syncing - increasing the consistency on disk, but decreasing performance.

To prevent buffers from skewing the results we decided to mount the all filesystems used for testing with the `sync` mount option. This requires every write to reach stable storage before completion.

3.3 The storage

The following performance graphs are measured using Iozone on the storage server. Appendix C contains the script used for the local storage test.

3.3.1 Iozone graphs

Figure 3.1 and 3.2 show respectively the read and write speeds measured on the storage server. The write graphs of all 4 filesystems show similar behaviour at the largest filesize and smallest record sizes. The cause of this is the configuration of the iozone test. During the larger file sizes, the smallest record sizes are skipped unless specified otherwise. Testing all larger file sizes with the smallest record sizes would take longer to test, without much added value.

3.3.2 Bonnie++ results

The reference metrics for Bonnie++ are presented in table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Read</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/sec</td>
<td>%CP</td>
<td>latency</td>
</tr>
<tr>
<td>ext3</td>
<td>Sequential</td>
<td>761</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>753</td>
<td>4</td>
</tr>
<tr>
<td>ext4</td>
<td>Sequential</td>
<td>808</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>757</td>
<td>5</td>
</tr>
<tr>
<td>xfs</td>
<td>Sequential</td>
<td>98</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>btrfs</td>
<td>Sequential</td>
<td>7212</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>7980</td>
<td>92</td>
</tr>
</tbody>
</table>

Table 3.1: Baseline Bonnie++ results

3.3.3 Interpretation

We can see that the read operations definitely benefit from fast cache access, where the speed only decreases on large operations that are not or only partially cached. Because of these caches there is very little difference between the filesystems, except for a small write hole in btrfs.

The write performance presents us with more interesting metrics, mostly due to the filesystem semantics and how they handle consistency and integrity. Ext3 and its counterpart ext4 behave very similar; they perform better when writing bigger record sizes and reach a maximum between 70 and 80 megabyte per second. XFS and btrfs are written for different purposes and also show
similar characteristics. XFS has been designed with consistent and predictable throughput in mind, a feature that clearly shows up in the graph. Btrfs is a GPL counterpart to the popular filesystem ZFS and is written with consistency in mind. Write performance is clearly lacking in raw throughput, but is regained at inode level; creating and deleting lots of files is something btrfs excels at.

Figure 3.1: Baseline read speeds


3.4 The network

The following sections show the measured round-trip time and bandwidth from the private cloud and public cloud. The round-trip time is measured using ping for the complete duration of the measurements of each filesystem type. The bandwidth is measured using iperf before and after each filesystem type.

3.4.1 Private cloud

Table 3.2 shows the round-trip time and bandwidth measurements for the private cloud. This is measured on a local reliable link. The bandwidth values clearly show the Gigabit Ethernet Card bottleneck. We consider these values consistent enough to get reliable filesystem performance results.
3.4.2 Public cloud

Table 3.3 shows the round-trip time and bandwidth measurements for the private cloud. The average bandwidth measured is 261Mbits/sec and a standard deviation of 24. The measured round-trip times are consistent enough as well. The deviation is higher compared to the private cloud, but still acceptable.

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>RTT min</th>
<th>RTT avg</th>
<th>RTT max</th>
<th>RTT mdev</th>
<th>Bandwidth before</th>
<th>Bandwidth after</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext3</td>
<td>0.166ms</td>
<td>0.439ms</td>
<td>11.017ms</td>
<td>0.998ms</td>
<td>943 Mbits/sec</td>
<td>940 Mbits/sec</td>
</tr>
<tr>
<td>ext4</td>
<td>0.168ms</td>
<td>0.445ms</td>
<td>8.895ms</td>
<td>0.991ms</td>
<td>943 Mbits/sec</td>
<td>942 Mbits/sec</td>
</tr>
<tr>
<td>xfs</td>
<td>0.166ms</td>
<td>0.452ms</td>
<td>12.391ms</td>
<td>0.988ms</td>
<td>942 Mbits/sec</td>
<td>940 Mbits/sec</td>
</tr>
<tr>
<td>btrfs</td>
<td>0.169ms</td>
<td>0.322ms</td>
<td>24.000ms</td>
<td>0.541ms</td>
<td>942 Mbits/sec</td>
<td>942 Mbits/sec</td>
</tr>
</tbody>
</table>

Table 3.3: Public cloud round-trip time and bandwidth
Chapter 4

Test results

4.1 Public cloud

4.1.1 Iozone graphs

Figure 4.1 shows the write speeds from the public cloud. The general write speeds are nearly 10% of the local write speeds. Looking at the values in table 3.3, the available bandwidth has enough room for roughly 5 times the measured write speeds. But it’s probably the latency of the link that limits the write performances on the filesystem. Between the different filesystems there’s not much difference, the shapes of the graphs show similar behavior among the filesystems. At the large file and record sizes, ext4 and btrfs are performing slightly better with 1-1.5 MBytes/sec more then respectively xfs and ext3.

4.1.2 Bonnie++ results

Table 4.1 shows the results from Bonnie++ run from the public cloud. Due to the low read and delete performance we ran the test with less files the reference metrics, 8 * 1024 files files. Since the read performance is affected by caching, bonnie++ wasn’t able to provide reliable results. This happens when the total time, in this case for the read speeds, is below 0.5 seconds. This doesn’t result in an actual value for the read performance, but we can conduct that the read speed is at least 8*1024*2 = 16384 files/sec.

We see similar results compared to the reference metrics, the create and delete values of btrfs are much higher then the other filesystems. The actual values however aren’t very impressive and could be a real bottleneck for many applications.

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Read</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/sec</td>
<td>%CP</td>
<td>latency</td>
</tr>
<tr>
<td>ext3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>14</td>
<td>0</td>
<td>181ms</td>
</tr>
<tr>
<td>Random</td>
<td>14</td>
<td>0</td>
<td>481ms</td>
</tr>
<tr>
<td>ext4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>14</td>
<td>0</td>
<td>273ms</td>
</tr>
<tr>
<td>Random</td>
<td>14</td>
<td>0</td>
<td>284ms</td>
</tr>
<tr>
<td>xfs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>34</td>
<td>0</td>
<td>89570µs</td>
</tr>
<tr>
<td>Random</td>
<td>34</td>
<td>0</td>
<td>273ms</td>
</tr>
<tr>
<td>btrfs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>3626</td>
<td>8</td>
<td>202µs</td>
</tr>
<tr>
<td>Random</td>
<td>3859</td>
<td>9</td>
<td>289µs</td>
</tr>
</tbody>
</table>

Table 4.1: Public cloud Bonnie++ results
4.2 Private cloud

4.2.1 Iozone graphs

Figure 4.2 shows the write speeds from the private cloud. The overall performance is about 4 times higher than the public cloud, but still not close to the reference metrics with a top of 22-25MBytes/sec. Again similar shapes among the filesystems, all reaching 25MBytes/sec, except xfs with 22MBytes/sec. If we compare this to the measured bandwidth in table 3.2 we see similar behavior as private cloud. With a top of 25MBytes/sec the private cloud is also using roughly 20% of the available bandwidth. The round-trip time between the private cloud and the storage server is around 0.4ms nothing compared to the public cloud, but still quite a lot compared with the reference metrics where the storage is attached locally.

4.2.2 Bonnie++ results

Table 4.2 shows the Bonnie++ results measured from the private cloud. Same as the other tests, btrfs is on top with the create and delete speeds. Interesting to note is that the number of files created and deleted per seconds is around 5 times as high as measured on the local storage server.
CHAPTER 4. TEST RESULTS

We weren’t able to locate the exact reason for this, we think that some network caching is taking place that isn’t used on the storage server itself.

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Read</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/sec</td>
<td>%CP</td>
<td>latency</td>
</tr>
<tr>
<td>ext3</td>
<td>Sequential</td>
<td>294</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>293</td>
<td>2</td>
</tr>
<tr>
<td>ext4</td>
<td>Sequential</td>
<td>293</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>294</td>
<td>2</td>
</tr>
<tr>
<td>xfs</td>
<td>Sequential</td>
<td>553</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>560</td>
<td>8</td>
</tr>
<tr>
<td>btrfs</td>
<td>Sequential</td>
<td>9285</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>9400</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 4.2: Private cloud Bonnie++ results
Chapter 5

Conclusion

We were able to build an initial ramdisk and use this to boot a cloud instance from a remote iSCSI target over a secure connection. We can conclude that as long as the environment is manageable to a certain extent (require the ability to modify block devices’ names) that this works as expected. The write performance presented earlier clearly shows an impact on the performance compared to the baseline. Nevertheless we still think that remote booting from the cloud is possible. Although we don’t have any data on what could be considered a community cloud, we think that this is the most usable environment. We noticed that the introduction of a few milliseconds of latency doesn’t influence the benchmark significantly. Depending on the application(s) used, the performance impact isn’t necessarily a problem and the community cloud also has some interesting use cases as presented in the introduction. Remote booting from the public cloud has a few unpreventable problems. The high latency has a big impact on the performance. Also the fact that long distance links are generally unreliable causes problems with iSCSI, since machines can only cope with a short interruption of service for their disks. After a timeout of less than 2 minutes the system simply crashes.

Research questions

How do we build a secure link to the storage system in the bootstrap phase of the cloud machine?

We were able to implement an IPsec tunnel from the init ramdisk phase, before mounting the rootdisk over iSCSI. There were minor problems assembling the initial ramdisk, but they were overcome rather quickly.

What filesystems are more suitable when used over iSCSI?

We did not notice significant differences between filesystems when used over a iSCSI connection on a local network. Therefore we recommend to go for a filesystem with high consistency guarantees like btrfs. Due to the unstable nature of TCP connections and possibly long recovery times of iSCSI sessions this is a good practice anyway.

What difficulties are involved with storage over the internet?

We found several issues:

- Higher latency, more jitter due to buffers in every layer;
• Unreliable connections;
• Security;
• Incompatible interfaces.

The high latency definitely impacts performance. Disk throughput is regularly slower than network throughput by a factor of 5. Note that this is without any tweaking and disabling the write buffers.

Unreliable connections make machines stall and eventually lose their disk. This is a significant problem due to potential loss of data, and a contributing factor to our choice for running the tests with the `sync` mount option.

We feel like we’ve provided adequate protection on our iSCSI Target server; however we’re still exposing our storage server to the internet. Organisations that really value their data might want to close exclusive deals with cloud providers for their own VPN connections or dedicated lines. This also provides the added benefit of being able to prevent a lot of the performance problems we’ve noticed.

The Amazon EC2 cloud did not allow us to bootstrap from an iSCSI device due to the throwing of a kernel panic upon initialization of the device (registration of the major/minor numbers in the kernel). There were collisions while trying to name the device which did not result in a panic post-bootstrap, but still required some workarounds before we could access the device.

**What are the security considerations involved with this solution?**

While our method prevents permanent storage of the data in the cloud, cloud providers still have access to the data *in transit*. Even though we’re using an IPsec connection, the cloud provider can access the data because they have access to the private key of the client in the initial ramdisk and in memory. In essence: this system does not help you if you distrust your cloud provider. Of course you will be able to monitor the disk usage more closely, but at that point it’s already too late.

Each client requires a custom ramdisk file. Management of all these bootstrap files has to be automated as to prevent user mistakes.

### 5.1 Future work

A few interesting topics for future research:

- We were unable to collect data for a community cloud environment. It would be interesting to see if the performance over longer, but fast and dedicated links is enough to provide a real usecase.
- Our results are focused on writespeeds and can be improved by enabling buffers and caches. Reading data is still a topic to look at.
- We only considered ext3, ext4, xfs and btrfs over iSCSI. An alternative is NFS, Network File System, a protocol designed to operate over a network. It runs on IP, thus can also be tunneled using IPsec between a cloud instance and a storage server.
- Our results are based on the chosen default parameters of IPsec and iSCSI. We have not looked at the impact of IPsec encryption on the performance.
- We did not perform *any* performance tuning whatsoever, though some filesystems allow for a lot of tuning options. For instance, disabling write barriers is expected to result in significant performance improvements (at the loss of consistency).
Appendix A

IPsec configuration files

The /etc/ipsec-tools.conf file is used by setkey from ipsec-tools to set the Security Policies in the kernel. The /etc/racoon/racoon.conf file is the configuration for racoon, the IKE daemon. Note that the following configuration files are from the perspective of the storage server.

A.1 /etc/ipsec-tools.conf

```
#!/usr/sbin/setkey -f

## Flush the SAD and SPD
flush;
spdflush;

# Always encrypt iSCSI to storage server
spdadd 145.100.104.20[3260] 0.0.0.0/0 any -P out ipsec esp/transport//require;
spdadd 0.0.0.0/0 145.100.104.20[3260] any -P in ipsec esp/transport//require;
```

A.2 /etc/racoon/racoon.conf

```
path pre_shared_key "/etc/racoon/psk.txt";

listen {
  adminsock disabled;
}

remote anonymous {
  exchange_mode aggressive;
  proposal_check strict;
  generate_policy on;
  nat_traversal on;
  dpd_delay 20;
  ike_frag on;
  proposal {
    encryption_algorithm aes;
    hash_algorithm sha1;
    authentication_method pre_shared_key;
    dh_group 2;
  }
}

sainfo anonymous {
  pfs_group 2;
  lifetime time 1 hour;
  encryption_algorithm aes;
  authentication_algorithm hmac_sha1;
  compression_algorithm deflate;
}```
Appendix B

Test script

```bash
#!/bin/bash

install-packages() {
  aptitude update
  aptitude -y install racoon ipsec-tools bonnie iperf open-iscsi kpartx lvm2
  xfsprogs btrfs-tools build-essential
}

ipsec() {
  service racoon restart
  service setkey restart
}

iscsi() {
  iscsiadm --mode discovery --type sendtargets --portal 145.100.104.20
  iscsiadm --mode node --targetname iqn.2011-02.nl.os3.studlab.paras:cloudinstance
                      --portal 145.100.104.20:3260 --login

  sleep 5
}

# HAX
kpartx -a /dev/sda
vgscan

filesystem() {
  fs=$1
  mkdir /mnt/iscsi
  case $fs in
  ext3)
    mkfs.ext3 /dev/cloudinstance/test
    ;;
  ext4)
    mkfs.ext4 /dev/cloudinstance/test
    ;;
  xfs)
    mkfs.xfs /dev/cloudinstance/test
    ;;
  btrfs)
    mkfs.btrfs /dev/cloudinstance/test
    ;;
  *)
    exit 1
  esac
  mount /dev/cloudinstance/test /mnt/iscsi -o sync
}

setup() {
  wget -O /mnt/work/iozone.tar http://www.iozone.org/src/current/iozone3.373.tar
  tar xf iozone.tar
  cd iozone3.373/src/current
  make linux-AMD64
}
```
APPENDIX B. TEST SCRIPT

```
files() {
  cat > /mnt/work/id_rsa <<EOF
  ...
EOF
  cat > /etc/ipsec-tools.conf <<EOF
  ...
EOF
  cat > /etc/racoon/racoon.conf <<EOF
  ...
EOF
  cat > /etc/racoon/psk.txt <<EOF
  ...
EOF
  cp /usr/share/doc/racoon/examples/samples/roadwarrior/client/phase1-* /etc/racoon/
}

perftest() {
  fs=$1
  stamp=${fs}.${(hostname).${[2]}
  mkdir /mnt/results
  mkdir /mnt/iscsi/iozone
  mkdir /mnt/iscsi/bonnie
  nohup ping -i 0.1 145.100.104.20 > /mnt/results/results.ping.${stamp} &
  sleep 10
  ./iozone -a -g 4096 -f /mnt/iscsi/iozone/io | tee /mnt/results/results.iozone.${stamp}
  chown 1000 /mnt/iscsi/bonnie
  bonnie++ -d /mnt/iscsi/bonnie/ -u 1000 -n 2 -s 0 | tee /mnt/results/results.bonnie.${stamp}
  sleep 10
  killall -2 ping
  scp -i /mnt/work/id_rsa /mnt/results/* perfresult@paras.studlab.os3.nl:
}

cleanup() {
  umount /mnt/iscsi
  dd if=/dev/zero of=/dev/cloudinstance/test bs=1M count=1
}

installpackages
files
ipsec
iscsi
setup

for FS in ext3 ext4 xfs btrfs; do
  filesystem 3FS $timestamp
  perftest $FS $timestamp
  cleanup
done
```
Appendix C

Reference metrics test script

```bash
#!/bin/bash

install_packages() {
    aptitude update
    aptitude -y install btrfs-tools xfsprogs
}

filesystem() {
    fs=$1
    mkdir /mnt/localdisk
    case $fs in
        ext3)
            mkfs.ext3 /dev/iscsi/localtest
        ;;
        ext4)
            mkfs.ext4 /dev/iscsi/localtest
        ;;
        xfs)
            mkfs.xfs /dev/iscsi/localtest
        ;;
        btrfs)
            mkfs.btrfs /dev/iscsi/localtest
        ;;
        *)
            exit 1
        ;;
    esac
    mount /dev/iscsi/localtest /mnt/localdisk -o sync
}

setup() {
    wget -O /mnt/work/iozone.tar http://www.iozone.org/src/current/iozone3.373.tar
    tar xf iozone.tar
    cd iozone3.373/src/current
    make linux-AMD64
}

perf_test() {
    fs=$1
    stamp=${fs}.$(hostname).$(date +%s
    mkdir /mnt/results
    mkdir /mnt/localdisk/iozone
    mkdir /mnt/localdisk/bonnie
    nohup ping -i 0.1 145.100.104.20 > /mnt/results/ping.$(stamp) &
    sleep 10
    ./iozone -a -g 4096 -f /mnt/localdisk/iozone/io | tee /mnt/results/results.iozone.$(stamp)
    chown 1000 /mnt/localdisk/bonnie
    bonnie++ -d /mnt/localdisk/bonnie/ -u 1000 -n 2 -s 0 | tee /mnt/results/results.bonnie.$(stamp)
```

APPENDIX C. REFERENCE METRICS TEST SCRIPT

```bash
sleep 10
killall -2 ping
scp -i /mnt/work/id_rsa /mnt/results/* perfresult@paras.studlab.os3.nl:
}
cleanup()
{
    umount /mnt/localdisk
    dd if=/dev/zero of=/dev/iscsi/localtest bs=1M count=1
}
installpackages
setup
timestamp=$(date +%F.%H%M)
for FS in ext3 ext4 xfs btrfs; do
    filesystem $FS $timestamp
    perftest $FS $timestamp
    cleanup
done
```
Bibliography