High Performance Object Stores

Gabryel Mason-Williams[1,2] Dave Bond [1] Mark Basham [1,3]

[1]Diamond Light Source

[2]University of Plymouth

[3]Rosalind Franklin Institute

June 2019

Big Data

High Performance Object Stores

Gabryel Mason-Williams[1,2] Dave Bond [1] Mark Basham [1,3]

diamond

[1]Diamond Light Source

[2]University of Plymouth

[3]Rosalind Franklin Institute

June 2019

Hello I am Gabryel Mason-Williams, I study computer science at the university of Plymouth and have been working within Scientific Computing over the last 12 months. My supervisor is Dave Bond and co-supervisor is Mark Basham.

Contents

Contents

Contents

How is data generated

Figure: Eiger Detector

Figure: Eiger Detector

Diamond can be viewed as giant video camera filming a sample and it rotates on its axis. We do this using Fancy camera's known as detectors, now these camera's generate a incredible amount of unstructured data a second. Put to put the amount into perceptive, a Netflix movie your might stream for a evenings entertainment is about 7GB whereas our detectors are producing anywhere between 1GB/s to 40GB/s.

Diamond = Netflix?

Figure: Diamond

Figure: Netflix

Diamond = Netflix?

Figure: Diamond

Figure: Netflix

At Diamond we stream science instead of movies

Contents

WHAT IS IT?

WHAT IS IT?

It is a way of handling and dealing with digital data, so you can retain it either temporally or permanently. There are different mechanises to do this currently at Diamond use a file storage

Storage - File Storage

File Storage

- Hierarchical Structure
- Files, Folders, Sub-directories, Directories
- Good with structured data
- Filesystem

```
[crc99971@ws257 ~]$ tree Talk/
Talk/
├── Notes
│   ├── main.tex
│   └── note.tex
├── Presentation
│   ├── Chapters
│   │   ├── ch1.tex
│   │   ├── ch2.tex
│   │   ├── ch3.tex
│   │   └── ch4.tex
│   ├── Images
│   │   ├── 1.pdf
│   │   ├── 2.pdf
│   │   └── 3.pdf
│   └── main.tex
└── References
    ├── main.tex
    └── reference.tex

5 directories, 12 files
[crc99971@ws257 ~]$
```

Figure: File Storage

File Storage

- Hierarchical Structure
- Files, Folders, Sub-directories, Directories
- Good with structured data
- Filesystem

```
[crc99971@ws257 ~]$ tree Talk/
Talk/
├── Notes
│   ├── main.tex
│   └── note.tex
├── Presentation
│   ├── Chapters
│   │   ├── ch1.tex
│   │   ├── ch2.tex
│   │   ├── ch3.tex
│   │   └── ch4.tex
│   ├── Images
│   │   ├── 1.pdf
│   │   ├── 2.pdf
│   │   └── 3.pdf
│   └── main.tex
└── References
    ├── main.tex
    └── reference.tex

5 directories, 12 files
[crc99971@ws257 ~]$
```

Figure: File Storage

So what about unstructured data, i.e images

File Storage - Unstructured data

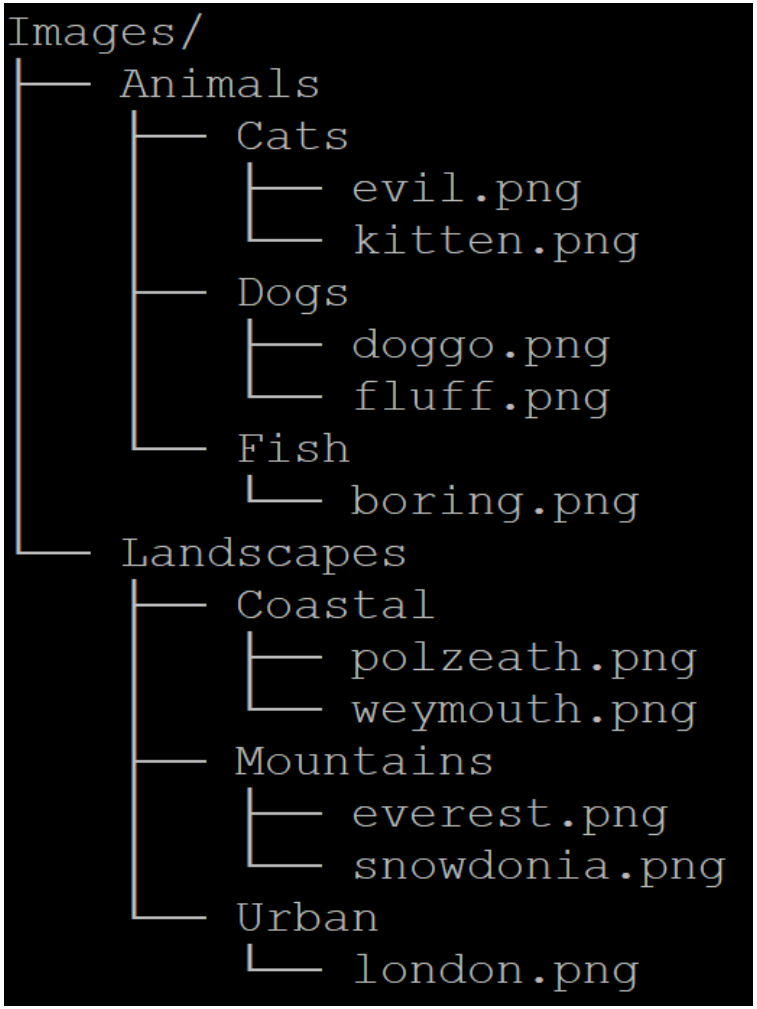


Figure: Image Folder

File Storage - Storing Images

```
Images/
├── Animals
│   ├── Cats
│   │   ├── evil.png
│   │   └── kitten.png
│   ├── Dogs
│   │   ├── doggo.png
│   │   └── fluff.png
│   └── Fish
│       └── boring.png
└── Landscapes
    ├── Coastal
    │   ├── polzeath.png
    │   └── weymouth.png
    ├── Mountains
    │   ├── everest.png
    │   └── snowdonia.png
    └── Urban
        └── london.png
```

Where to store

A Images/Animals/Dogs

B Images/Animals/Cats

C Images/Landscapes/Urban

File Storage - Storing Images

```
Images/
├── Animals
│   ├── Cats
│   │   ├── evil.png
│   │   └── kitten.png
│   ├── Dogs
│   │   ├── doggo.png
│   │   └── fluff.png
│   └── Fish
│       └── boring.png
└── Landscapes
    ├── Coastal
    │   ├── polzeath.png
    │   └── weymouth.png
    ├── Mountains
    │   ├── everest.png
    │   └── snowdonia.png
    └── Urban
        └── london.png
```

Where to store

A Images/Animals/Dogs
B Images/Animals/Cats
C Images/Landscapes/Urban

If we have a file-system for storing images, that has subdirectories, Landscapes and Animals, and say Landscapes has subdirectories of Coastal, Mountain, Urban, etc. and Animals has subdirectories of; Fish, Dogs, Cats, etc. Now if we have well-defined images, of just Dogs, Coasts, Mountains and Cats. This system works perfectly well.

File Storage - Storing Images

```
Images/
├── Animals
│   ├── Cats
│   │   ├── evil.png
│   │   └── kitten.png
│   ├── Dogs
│   │   ├── doggo.png
│   │   └── fluff.png
│   └── Fish
│       └── boring.png
└── Landscapes
    ├── Coastal
    │   ├── polzeath.png
    │   └── weymouth.png
    ├── Mountains
    │   ├── everest.png
    │   └── snowdonia.png
    └── Urban
        └── london.png
```

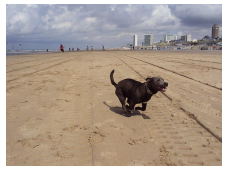
Where to store

A Images/Animals/Dogs

B Images/Landscapes/Coastal

C Images/Landscapes/Urban

File Storage - Storing Images



```
Images/
├── Animals
│   ├── Cats
│   │   ├── evil.png
│   │   └── kitten.png
│   ├── Dogs
│   │   ├── doggo.png
│   │   └── fluff.png
│   └── Fish
│       └── boring.png
└── Landscapes
    ├── Coastal
    │   ├── polzeath.png
    │   └── weymouth.png
    ├── Mountains
    │   ├── everest.png
    │   └── snowdonia.png
    └── Urban
        └── london.png
```

Where to store

A Images/Animals/Dogs
B Images/Landscapes/Coastal
C Images/Landscapes/Urban

However, if we have an image of a Dog at the Beach, where do you place this image within this structure. It could either go in the Dogs or Coastal directory either would make sense. However, when looking for that image again, are you going to have remembered where you put it. Object Storage would make sense here as the data doesn't have an inherent structure as much as we can try.

File Storage - Pros and Cons

Pros

- Stores data in Hierarchical structure
- Ideal for structured data
- Human interpretable
- Already in use at Diamond

Cons

- Not good with unstructured data
- File size issues
- Lots of files
- Not cloud native
- Expensive

Contents

Object Storage - What is it

Object Storage

- Archival data
- Stores data in flat structure
- Performant with unstructured data
- Flexible data sizes, very small/very large
- RESTful API

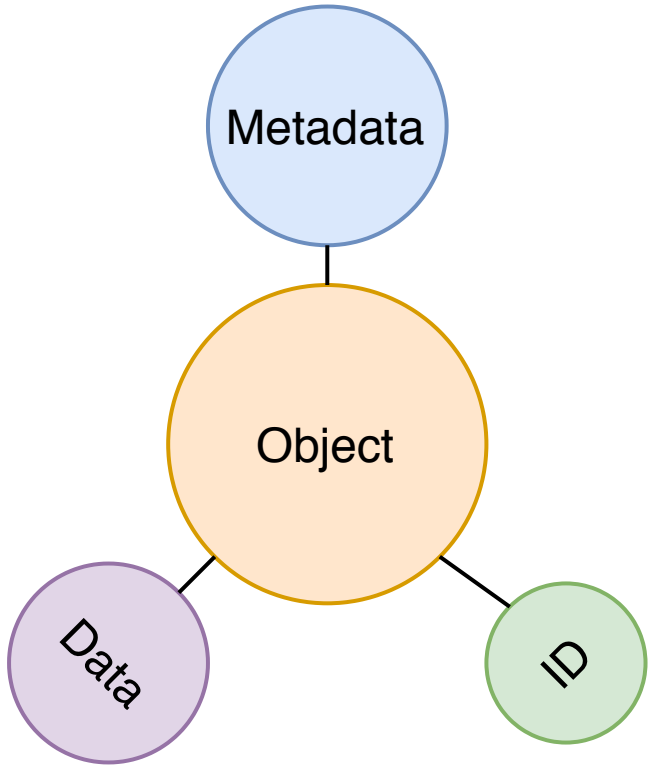


Figure: Object Structure

Object Storage - What is it

Object Storage

- Archival data
- Stores data in flat structure
- Performant with unstructured data
- Flexible data sizes, very small/very large
- RESTful API

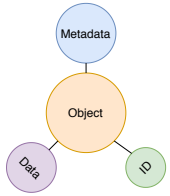


Figure: Object Structure

So this is a way to deal with unstructured data, in a sensible way that makes sense. Store the data as an object with a rich metadata that describes information about it, and associate an ID to it.

Object Storage - Image Example

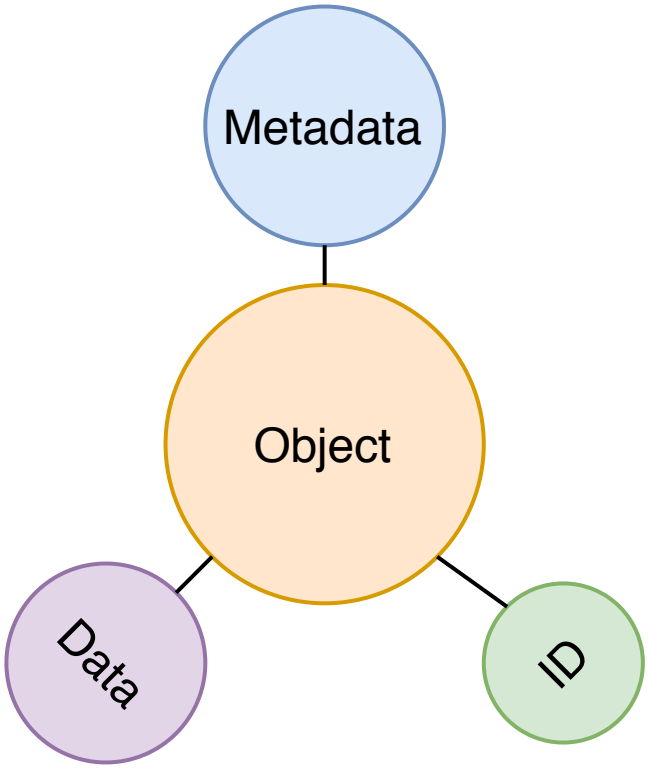


Figure: Object

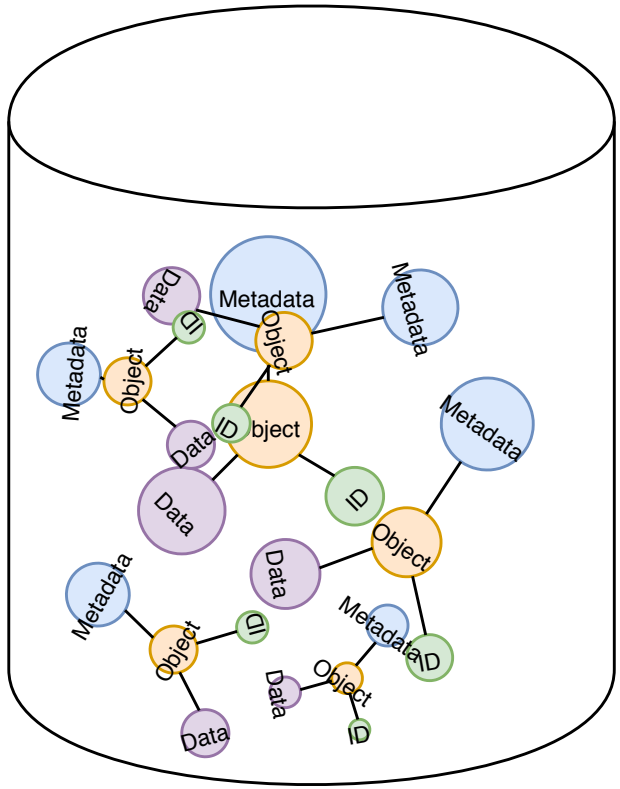


Figure: Image Pool

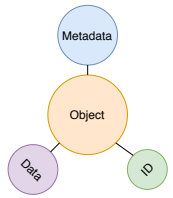


Figure: Object

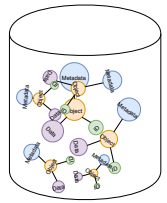


Figure: Image Pool

Instead, we can have a Pool called images and place the images inside that pool as objects, with metadata attached identifying what the image is instead. Now we can just put images in the pool without having to worry where to put them. And when we want to get the image with a Dog at the Beach, we can look through the metadata and find images with that included that information.

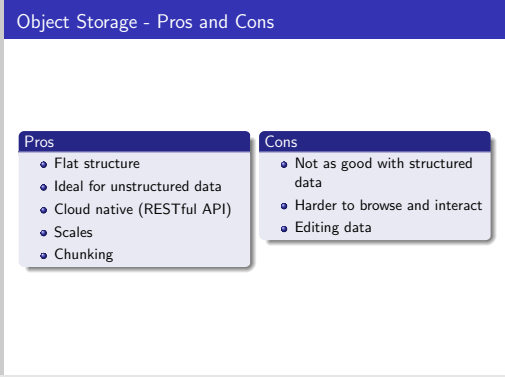
Object Storage - Pros and Cons

Pros

- Flat structure
- Ideal for unstructured data
- Cloud native (RESTful API)
- Scales
- Chunking

Cons

- Not as good with structured data
- Harder to browse and interact
- Editing data



So the flat structure makes the retrieval of data a lot easier and efficient, due to the way you access the data it is perfect for cloud integration. It scales easily with limited overhead, no servers dedicated to file structure. There are cons of course it is not a magic bullet to storage issues, most obvious is structure data, the fact it is not as easy to browse an object store and interact with it there are applications that help this problem however, DosNa being one, and you can't append to an object you need to read and write the whole thing.

Contents

Contents

Part 2

Catch up

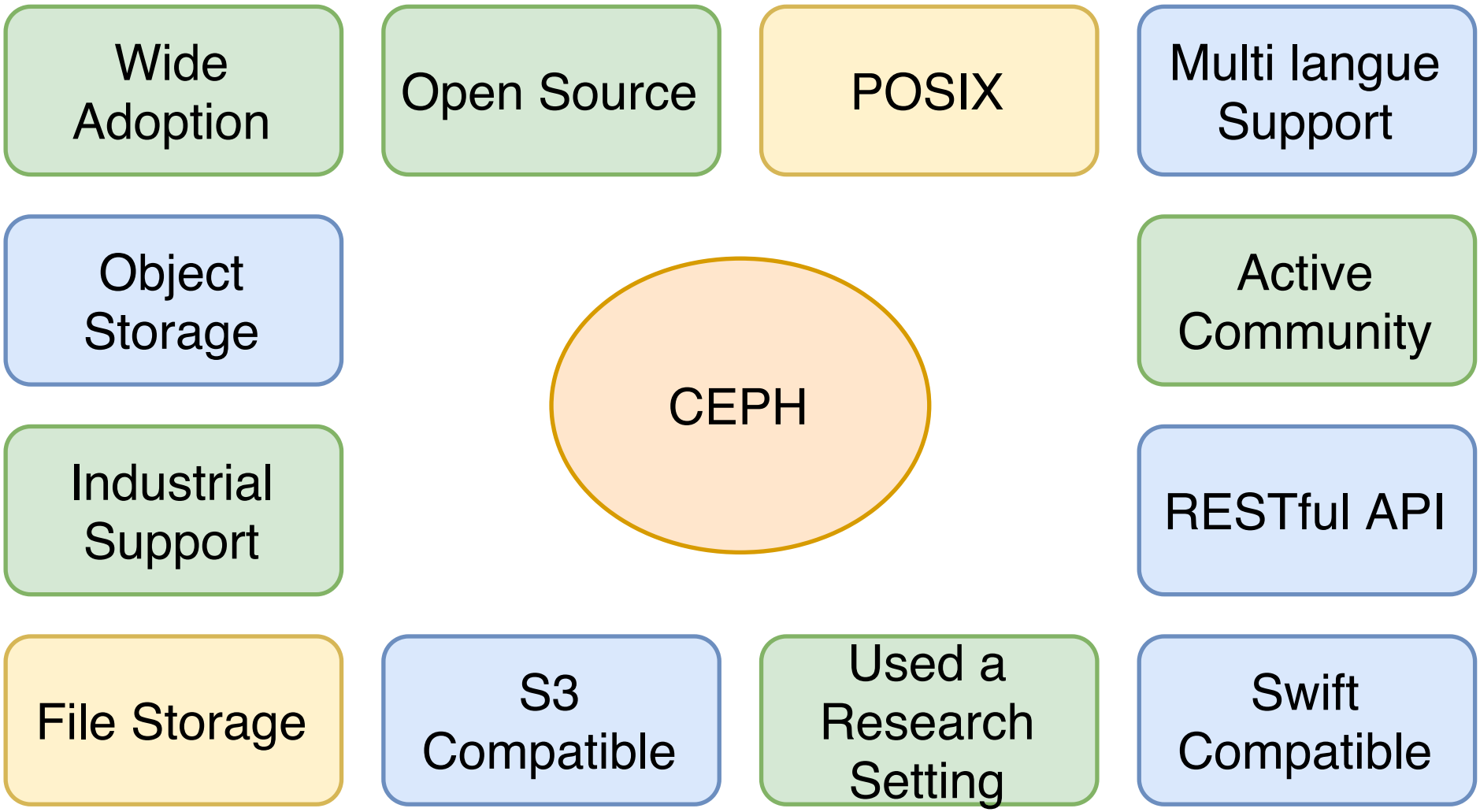
- Data at Diamond
- File Storage
- The problem with File Storage
- Object Storage and what it is

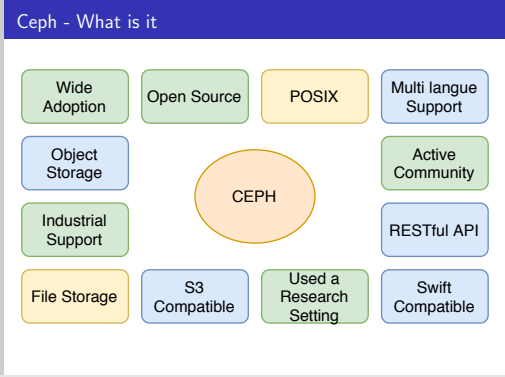
Where we are going

- Ceph
- RAM
- Storage

Contents

Ceph - What is it





So Ceph is essence is Open Source scale-able distributed storage platform, that be used as a POSIX File System, Object store or Block device. It is S3 and Swift Compatible and has a a restful API that is implemented in multiple languages including Python,C++ and Java. It is approximately 14 years old and started as PhD project however has since been acquired by Redhat. It has many industrial and research facilities that use it, including STFC, CERN, OCADO and BLOOMBERG

Ceph - Why is Diamond Interested

Reasons

- File & Object Storage
- Open Source
- Scales
- Cloud integration

Now you make wonder why are interested in it for File storage when we deal with unstructured data, that because there is a lot of change that would be required some of which is already happening before we would go to full object store.

Contents

RAM - What is it?

RAM

- Volatile
- Fast
- Low latency
- Expensive

Figure: Not this

RAM - Why we are interested

Reasons

Transient data

Fast

When dealing with Transient data it shouldn't be stored in main memory if it can be avoided as this slows down the process.

RAM - How to access it?

In a file system

- Mount it in a directory
- Use a temporary file system, i.e tmpfs
- mount -t tmpfs -o size=8G tmpfs /mnt/ram

In object storage (Ceph)

- Well its not as easy.

RAM - How to access it?

In a file system
- Mount it in a directory
- Use a temporary file system, i.e tmpfs
- mount -t tmpfs -o size=8G tmpfs /mnt/ram

In object storage (Ceph)
- Well its not as easy.

It is quite simple, you would create a directory such as /mnt/ram and then apply a temporary file system to it. An wallah you have access to RAM in a file storage way.

Contents

Ceph - How to expose storage

What the Computer sees

```
NAME                  MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                     8:0    0 894.3G  0 disk
sdb                     8:16   0 894.3G  0 disk
sdc                     8:32   0 222.6G  0 disk
├─sdc1                  8:33   0     1G  0 part /boot
├─sdc2                  8:34   0  25.9G  0 part
│ ├─VolGroup00-root   253:0    0  17.9G  0 lvm  /
│ └─VolGroup00-swap   253:1    0     8G  0 lvm  [SWAP]
└─sdc3                  8:35   0 195.7G  0 part
  └─DataGroup-data    253:2    0 195.7G  0 lvm  /exports
```

Figure: Hard drive - sda

What we see

Figure: Hard drive - sda

What the Computer sees

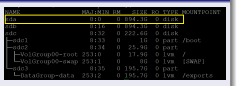


Figure: Hard drive - sda

What we see

Figure: Hard drive - sda

If we look here we can see that RAM is not exposed in the same way a hard drive is, this is an issue for us, as we want to be use RAM for storage

Can we make RAM = Disk?

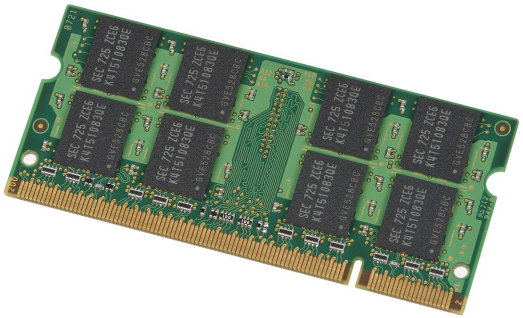


Figure: RAM Drive

Figure: Hard drive - sda

Figure: RAM Drive

Figure: Hard drive - sda

Well the answer is yes and it isn't as difficult as it may seem, in essence we can trick the Linux kernel, into thinking that the RAM drive is actually a block device. How do we do this? Well there are two techniques, you can make your own Linux Module to do it or use the default, or change a pre-existing one to meet your needs.

GRAM VS BRD

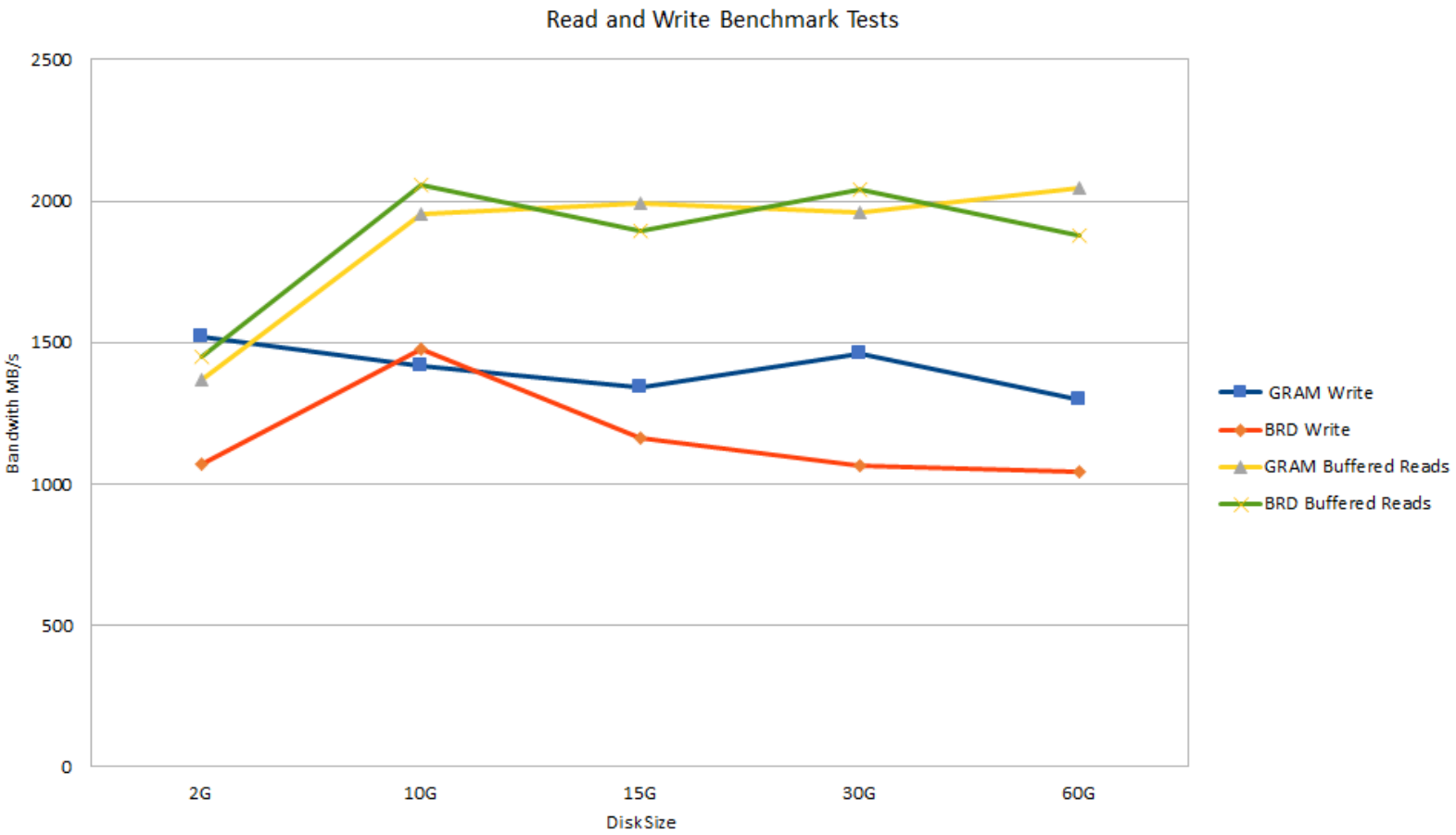


Figure: RAM Block Devices

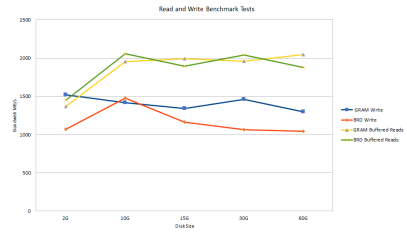


Figure: RAM Block Devices

So I should give a bit of background GRAM (General RAM) originally started off as my own kernel module, however the performance was to say for the least abysmal, i.e. the speed of a hard drive, due to the way I allocated memory and read and read from it. So after so serious googling I came across ZRAM (Which is a Compression RAM Block Device), so I removed the compression part and gave birth to GRAM. BRD is the default Linux provided block device, what is interesting to see is that BRD is less performant, that's a story for another day.

Contents

Part 2

Catch up

- What Ceph is
- What Ram is
- How to access RAM

Where we are going

- Distributed RAM Object Store
- Performance
- Use case
- Future Work and Conclusion

Contents

Compute Cluster - What is it

About

- Interconnected Computers
- Disk less (No Storage/Hard drives)
- Lots of RAM
- Powerful CPUs & GPUs

Figure: Actually this

About

- Interconnected Computers
- Disk less (No Storage/Hard drives)
- Lots of RAM
- Powerful CPUs & GPUs

Figure: Actually this

I won't go into too much detail here as James will be giving a talk next week on this, however, what this means is that we have a system that has a lot of RAM that we can use as storage for running our Ceph instance

Cluster - Deploying Ceph

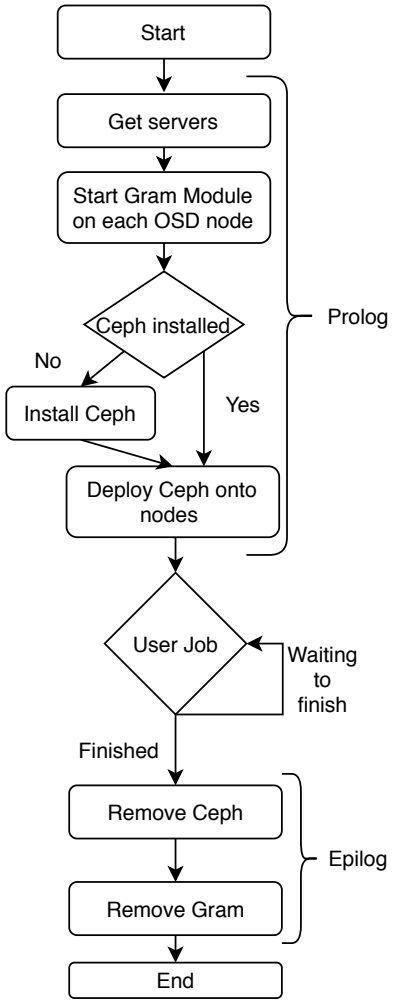


Figure: Deploy Script

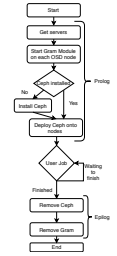
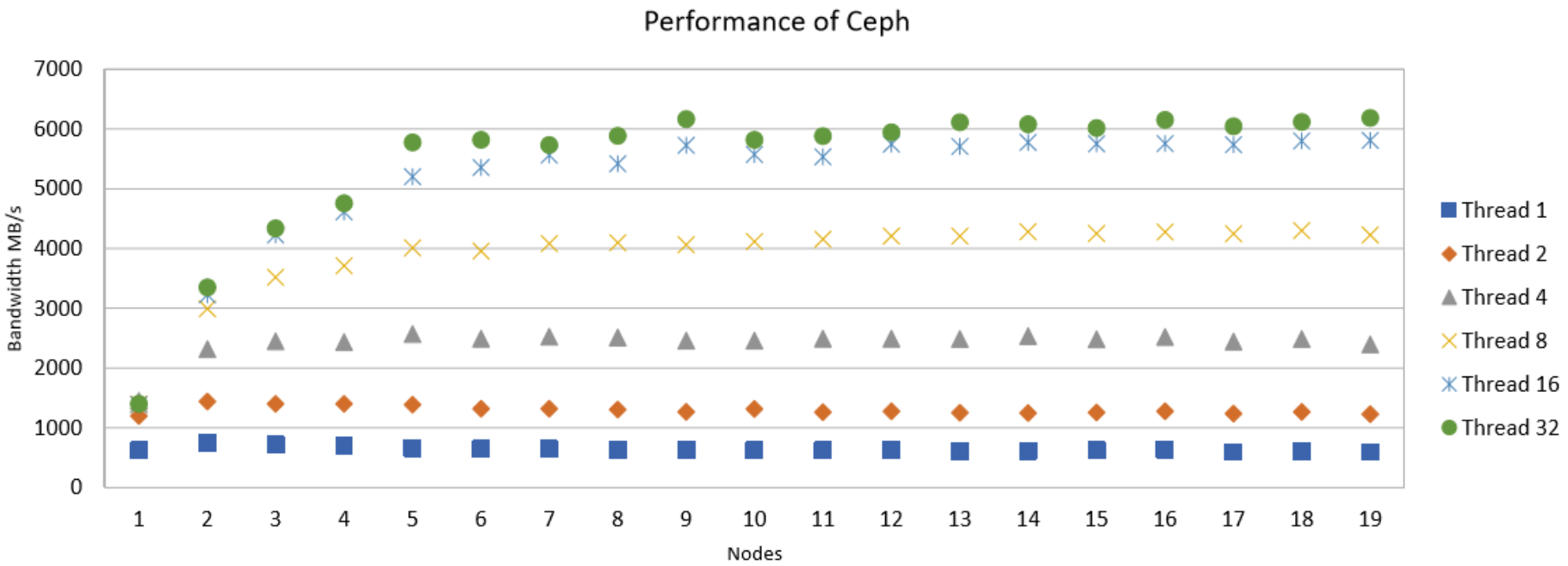


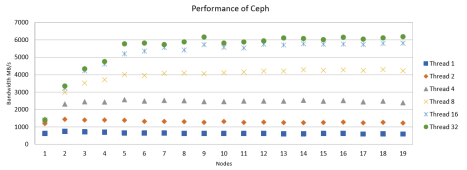
Figure: Deploy Script

Walk through Script

Contents

Performance - What we got





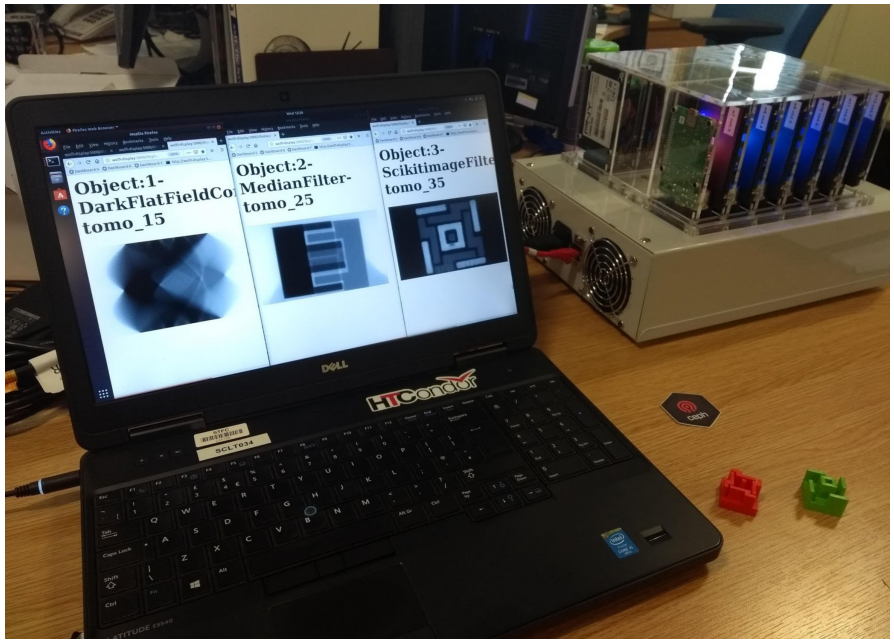
So the network limit is approximately 6GB/s, so we are able to reach this in 6 Nodes, which is very good to see, if run on a compute cluster with more per format RAM we could potentially see this number be a lot lower, as this cluster operates at approx 1GB/s per OSDs. Whats interesting to see here is that 16 threads should do the job just fine which means we can run a job at the same time creating a hyper-converged platform.

Contents

Distributed Ram Object Store - Use Case

Tomography data

- Savu
- Has an object store plugin DosNa
- See objects and data processing in real time
- Run irrespective of current File system performance



High Performance Object Store - Use Case

MX Pipeline

- Remove file system issues

Contents

Conclusion & Future Work

Contents

Questions

Thanks for listening any questions?

Contact details:
Slack: @Gabryel (Gabryel Mason-Williams)
Email: gabryel.mason-williams@diamond.ac.uk

Thanks for listening any questions?

Contact details:
Slack: @Gabryel (Gabryel Mason-Williams)
Email: gabryel.mason-williams@diamond.ac.uk

I would like to say thank you to my supervisors Dave Bond, Mark Basham, scientific computing, the ceph community and everyone who's compute time I took whilst running my tests