Analyzing CBT Benchmarks in Jupyter

Nov 16, 2016

Ben Lynch

Minnesota Supercomputing Institute

Background

- MSI has 1 production Ceph cluster used for Tier 2 storage.
- The Tier 2 storage is available exclusively through the S3 API.
- It has 3.2 PB of raw disk that we use with 4+2 erasure coding.
- Deployed running Firefly, now it's running Jewel

- MSI is deploying a new Ceph cluster as part of a new OpenStack cloud resource.
- We need to run some tests on our new hardware

Goals

Test the hardware and configuration for a new Ceph cluster

- Identify faulty hardware or configuration

Tune Ceph and system parameters to improve performance

- Number of placement groups, recovery threads, FileStore settings ...
- System network parameters
- TCMalloc memory usage

Test recovery time for various failures or system changes

- Single HDD failure
- Single SSD failure
- 12 HDD OSD failure
- Placement group split from 4096 to 8192 pgs

More Goals

Test less-supported features

- BlueStore

Collect base storage performance for reference when we move to application benchmarks on OpenStack VMs

- Bandwidth for SATA and SSD data pools
- IOPs and latency distribution for SATA and SSD pools

What is CBT?

Ceph Benchmark Tool

CBT is a python tool for configuring Ceph clusters and running tests.

- A passphraseless ssh key is setup for cbt
- cbt user is given sudo privilages
- CBT uses pdsh to
  - configure the monitor
  - configure the OSDs
  - format disks
  - run tests
  - copy test results back to server where cbt is running

What CBT does not do

Parsing

- CBT does not parse the output from the tests

What is Jupyter?

- Web-based environment for data analysis and more
- Evolved from iPython notebooks
- Run R, Python, or Julia

In [2]:

```
import cbtworkspace
```

We input the location for our cbt output.

In [3]:

```
cw = cbtworkspace.cbtWorkspace('/Users/blynch/Documents/Code/b
enchmarks/stratus')
```

In [4]:

```
cw.ls()
```

Out[4]:

```
['test1',
 'test10',
 'test100',
 'test101',
 'test102',
 'test103',
 'test104',
 'test105',
 'test11',
 'test12',
 'test13',
 'test14',
 'test15',
 'test16',
 'test17',
 'test18',
 'test19',
 'test20',
 'test21',
 'test22',
 'test23',
 'test24',
 'test25',
 'test98',
 'test99']
```

.ls() looks at all the subdirectories and determines which ones look like CBT output.

We can get a general description of a test

In [5]:

```
cw.describe('test13')
```

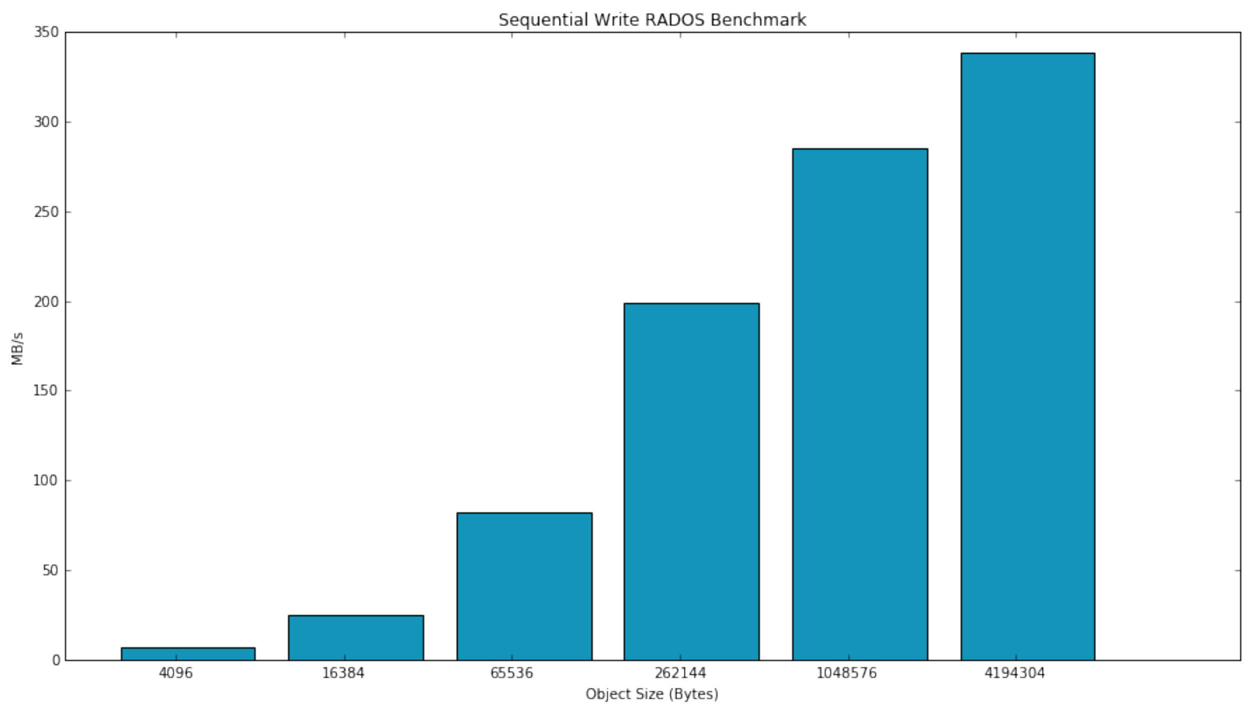
	Archive	Benchmark	Size	Test	Read	Write
0	test13	Radosbench	4096	seq	9.07837	0.00000
1	test13	Radosbench	4096	seq	9.20957	0.00000
2	test13	Radosbench	4096	seq	9.01341	0.00000
3	test13	Radosbench	4096	seq	8.78927	0.00000
4	test13	Radosbench	4096	write	0.00000	8.68023
5	test13	Radosbench	4096	write	0.00000	8.79946
6	test13	Radosbench	4096	write	0.00000	8.86850
7	test13	Radosbench	4096	write	0.00000	8.40140
8	test13	Radosbench	4194304	seq	365.91600	0.00000
9	test13	Radosbench	4194304	seq	370.25200	0.00000
10	test13	Radosbench	4194304	seq	367.47700	0.00000
11	test13	Radosbench	4194304	seq	369.61100	0.00000
12	test13	Radosbench	4194304	write	0.00000	489.35300
13	test13	Radosbench	4194304	write	0.00000	507.49200
14	test13	Radosbench	4194304	write	0.00000	497.59000
15	test13	Radosbench	4194304	write	0.00000	513.46000

Bandwidth for RADOS Write Benchmark on a single node (4k - 4M block sizes)

In [7]:

```
cw.bar_graph(['test4'],'rados_write',title='Sequential Write RADOS Benchmark',
             log=False, bar_label=False)
```

```
<matplotlib.figure.Figure at 0x113f23320>
```

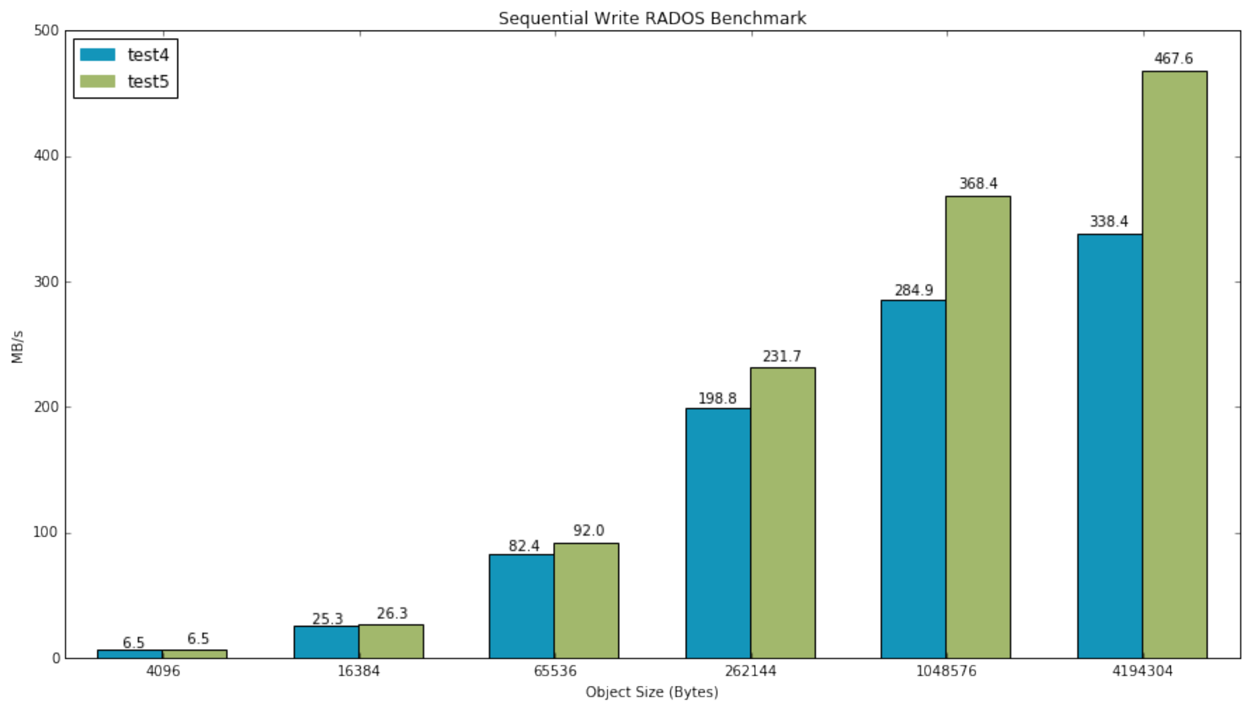


Comparison of a single storage node before and after proper partition alignment on the NVMe

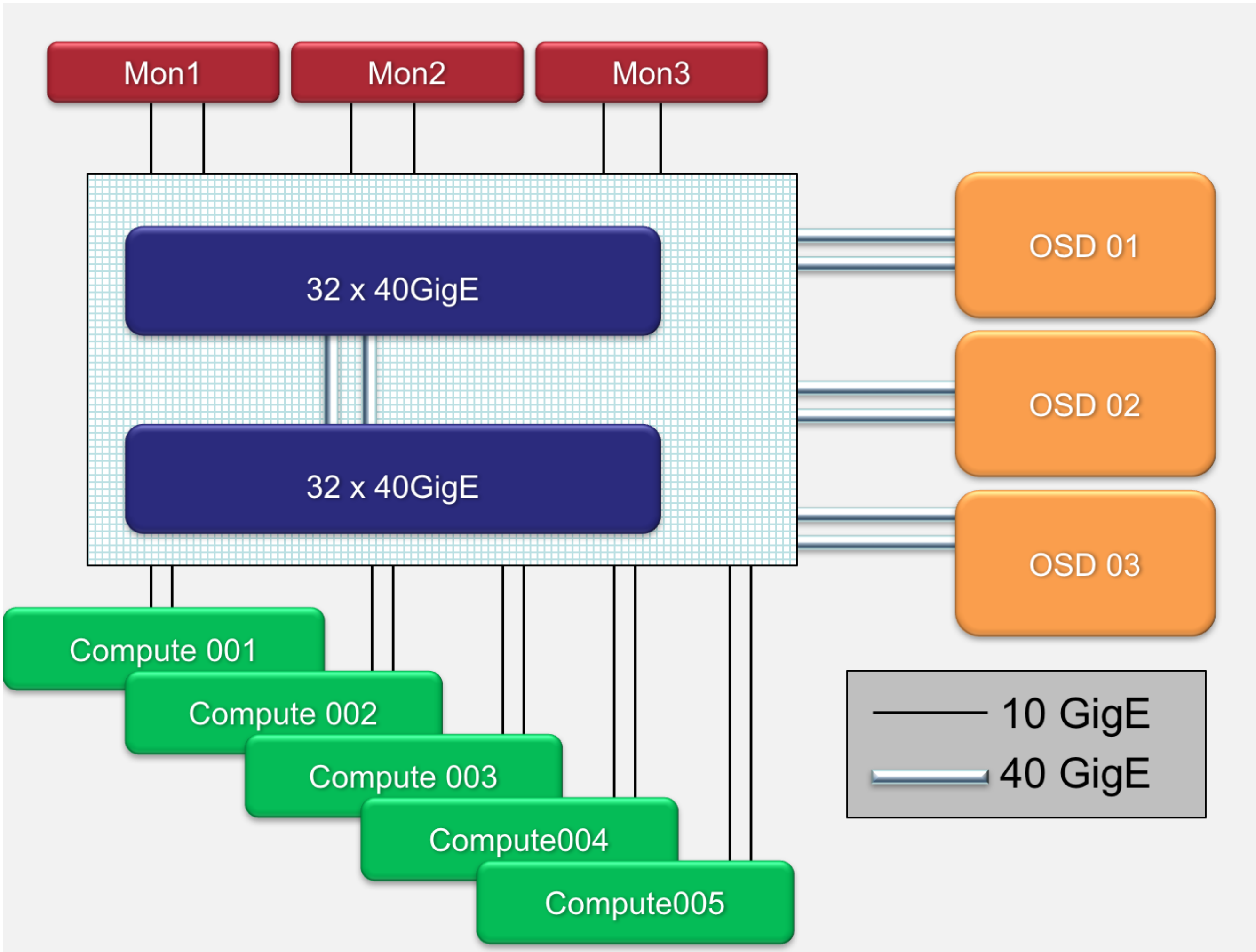
In [48]:

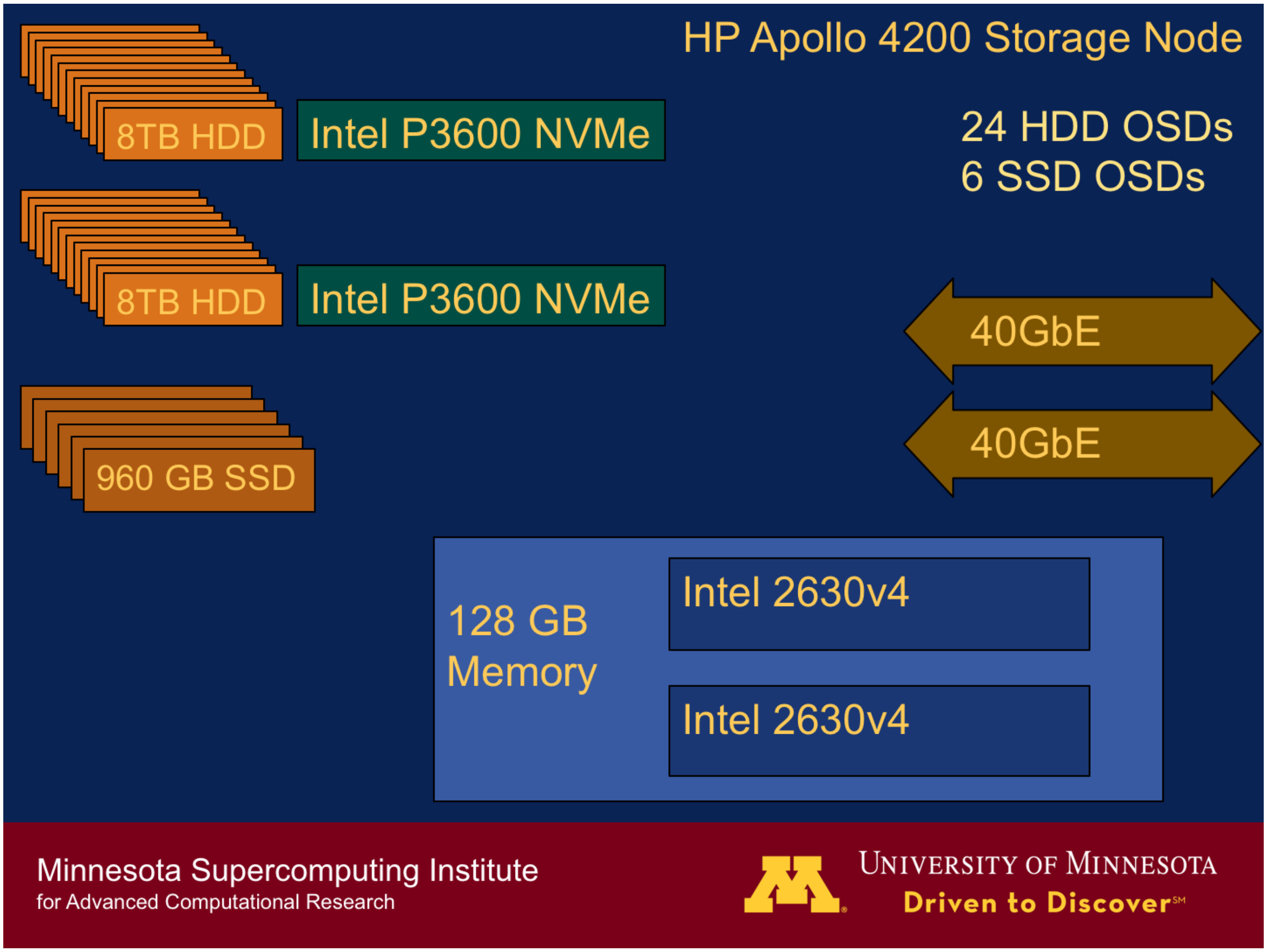
```
cw.bar_graph(['test4','test5'],'rados_write',
             title='Sequential Write RADOS Benchmark', log=False)
```

<matplotlib.figure.Figure at 0x1183c3470>



Hardware


Mon1
Mon2
Mon3
32 x 40GigE
32 x 40GigE
OSD 01
OSD 02
OSD 03
Compute 001
Compute 002
Compute 003
Compute004
Compute005
10 GigE
40 GigE


HP Apollo 4200 Storage Node
24 HDD OSDs
6 SSD OSDs
8TB HDD
Intel P3600 NVMe
8TB HDD
Intel P3600 NVMe
960 GB SSD
40GbE
40GbE
128 GB Memory
Intel 2630v4
Intel 2630v4
Minnesota Supercomputing Institute
for Advanced Computational Research
University of Minnesota
Driven to Discover℠

Endurance and Recovery Tests

Step 1. Fill cluster to ~50% full and test if it maintains reasonable write speeds

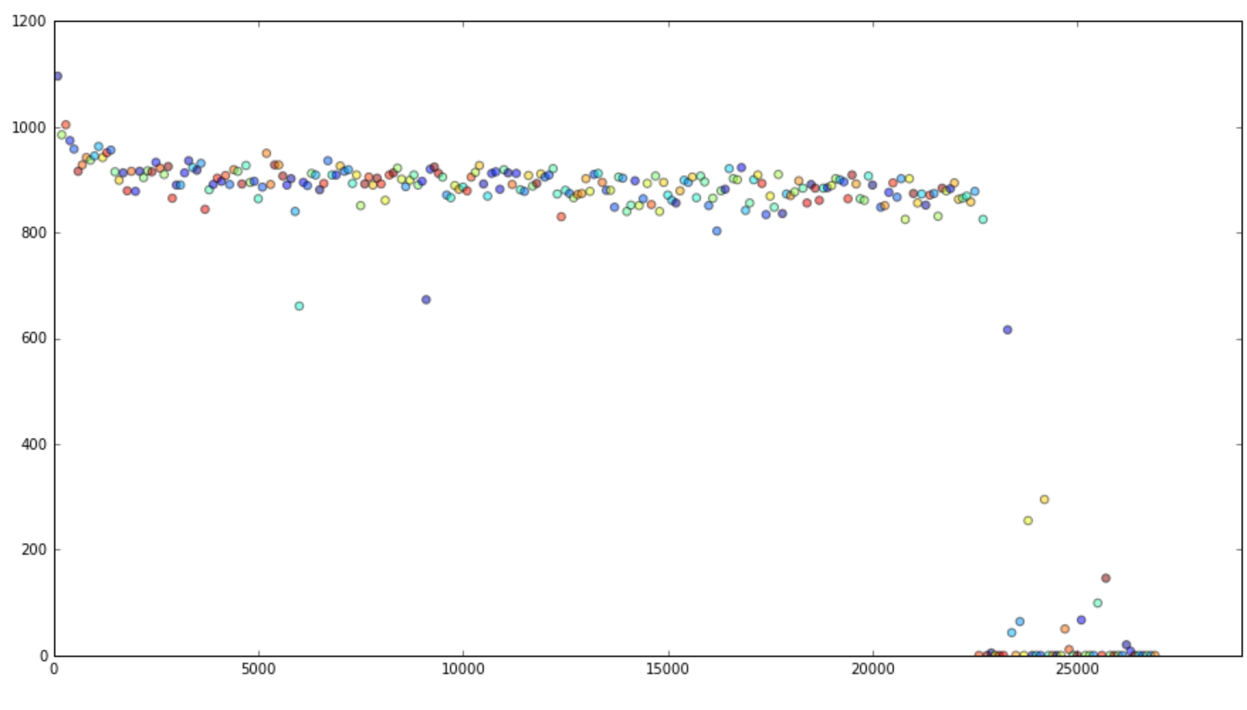
Step 2. Remove a disk and measure time to recover

In [27]:

```
plt.figure(num=None, figsize=(15, 8), dpi=80, facecolor='w', e
dgecolor='k')
plt.ylim((0,1200))
plt.xlim((0,29000))
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```

Out[27]:

```
<matplotlib.collections.PathCollection at 0x1177ce048>
```



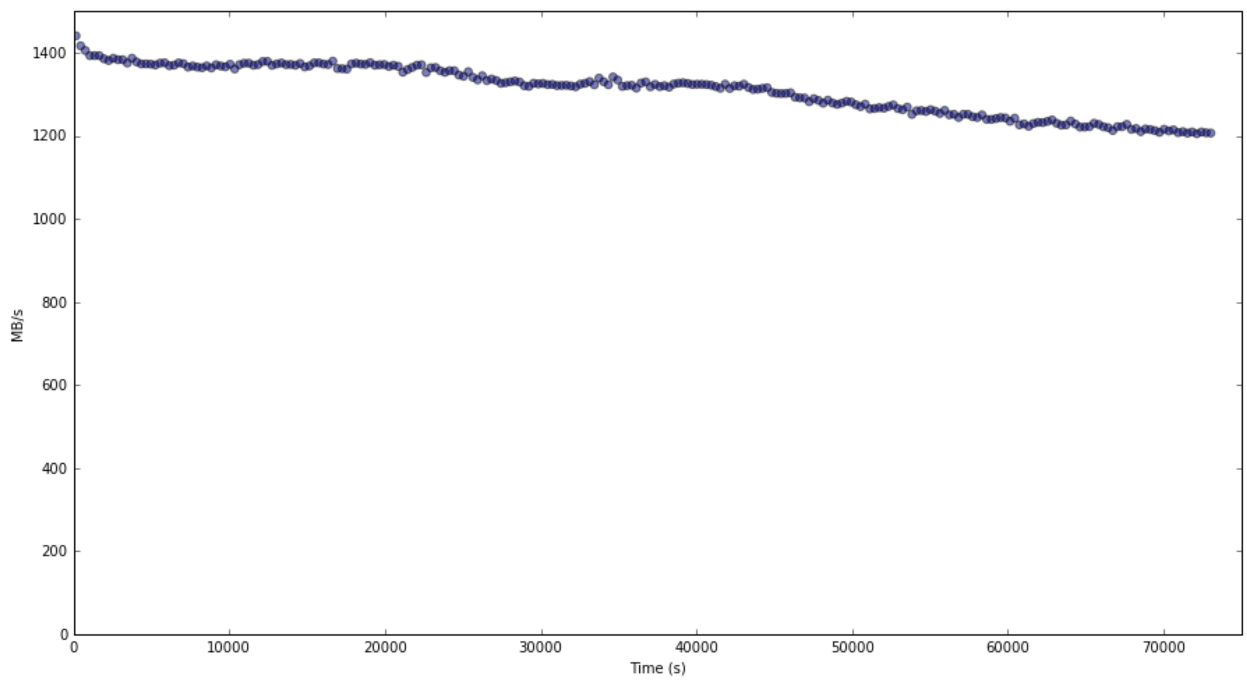
Filling the cluster to 50.9% in 20:17:30

In [31]:

```
plt.figure(num=None, figsize=(15, 8), dpi=80, facecolor='w', e
dgecolor='k')
plt.ylabel('MB/s')
plt.xlabel('Time (s)')
plt.ylim((0,1500))
plt.xlim((0,75000))
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```

Out[31]:

```
<matplotlib.collections.PathCollection at 0x116f62a58>
```

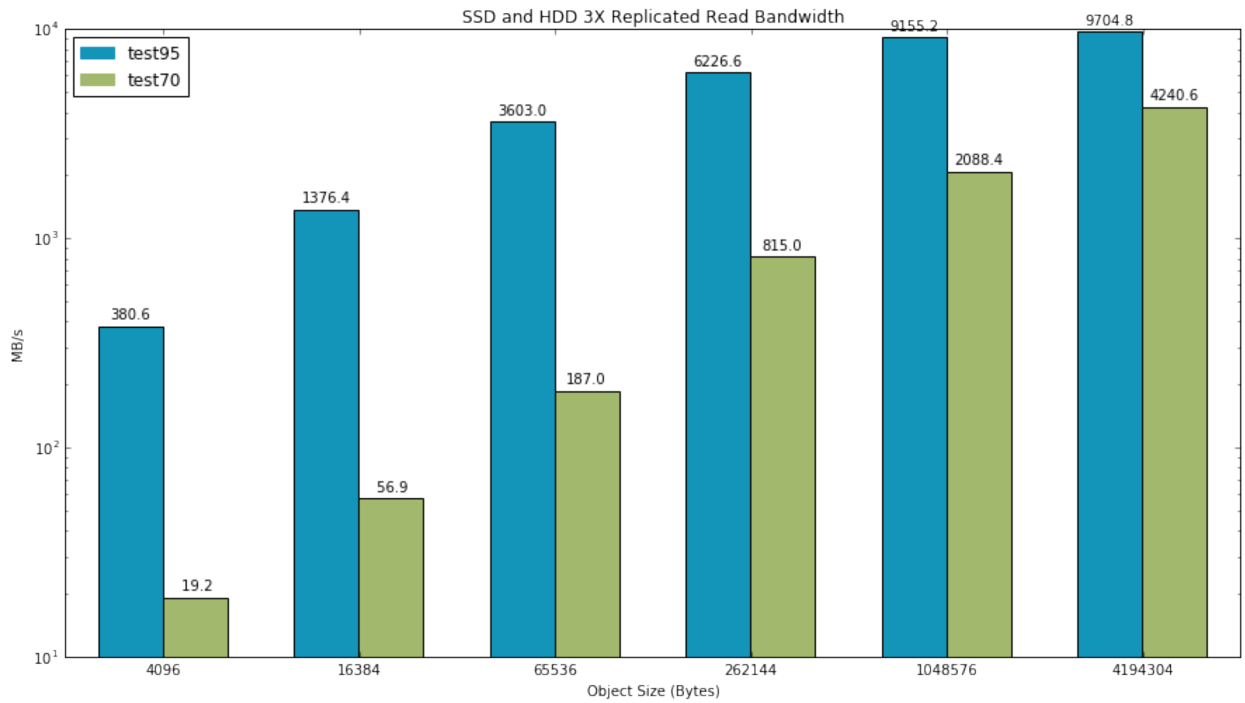


SSD and HDD 3X Replicated Read Bandwidth

In [41]:

```
cw.bar_graph(['test95','test70'],'rados_read_seq',title='SSD a
nd HDD 3X Replicated Read Bandwidth',
             log=True, bar_label=True)
```

<matplotlib.figure.Figure at 0x1183d7b70>



BlueStore

In [43]:

```
cw.bar_graph(['test101','test70'],'rados_write',title='BlueStore vs Fi
leStore Write Bandwidth',
             log=True, bar_label=True, legend_labels=['BlueStore', 'Fi
leStore'])
```

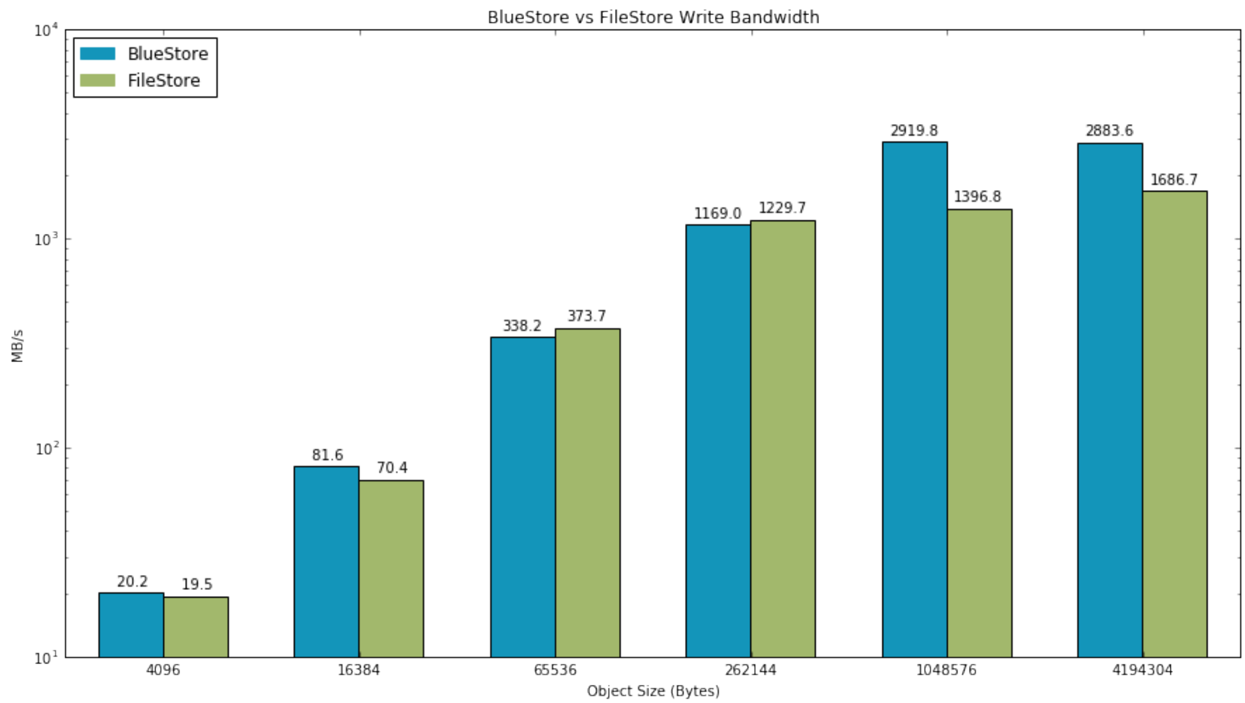
```
<matplotlib.figure.Figure at 0x117fd3630>
```

In [46]:

```
cw.geomean('test101','rados_write')/cw.geomean('test70','rados_write')
```

Out[46]:

```
1.2428060674586985
```



Acknowledgements

Graham Allan

Matt Mix