Hypervisor optimisation and benchmarking studies for running Grid Middleware services

Robert Currie

Contents

Hypervisor Description

Aim is to build a Hypervisor setup which gives us redundancy against hardware failure.

Hypervisor setup is designed to use ZFS as a software RAID(mirror) and use DRBD (Distributed Replicated Block Device) as a backup/replication technology.

We have 2 hardware Hypervisors for hardware failover, both with the same SL7 configuration:

- 32 Intel cores
- 128Gb RAM
- 2 x 1Tb disks for VMs
- 1 x dedicated 10Gb fiber link

HV Description

Using DRBD requires a block device in order to work (usually a hardware RAID or physical device).

ZFS is able to create virtual block devices known as ZVol. Here the write operations are mirrored across 2 disks in each hypervisor.

DRBD manages the ZVol block device then presents a new block device to the system. This new device mirrors all write operations at the block level over the network.

On top of the DRBD block device we make a standard ext4 filesystem and run our hypervisors on top of it.

HV Description (Storage)

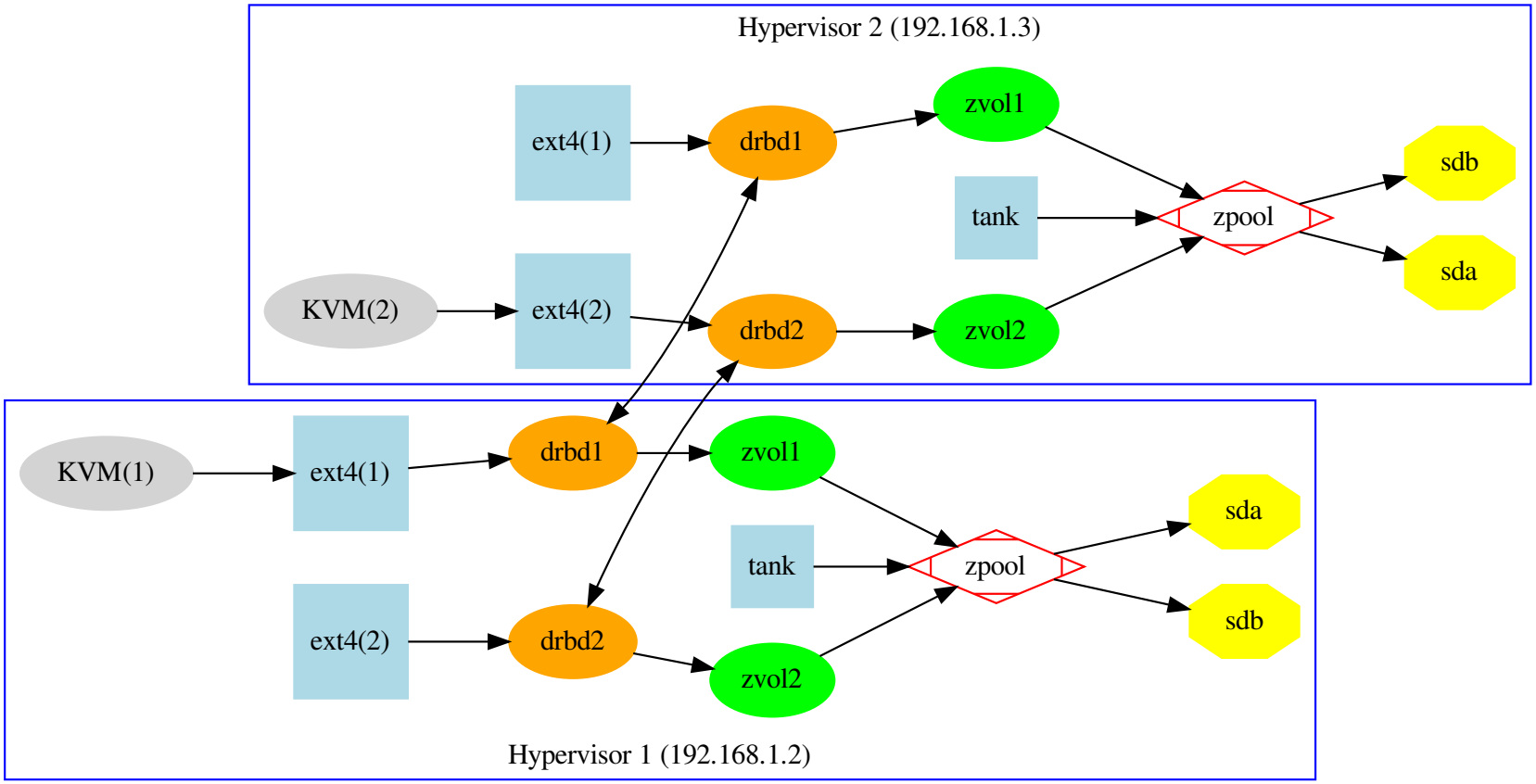


Diagram of the storage setup within the Hypervisors.

HV Benchmarking

Recently encountered some problems running a DPM head node VM on the hypervisor:

- Reading a large file (50Gb) was extremely slow (2-3Mb/s).
- All of the RAM (128Gb) had been consumed by caches for 1 VM (VM size $\approx$ 60Gb)
- Writing a large file from the Host caused I/O transactions on the guest to stop.
- Write operations on the Host caused very high load spikes on Host and Guests.

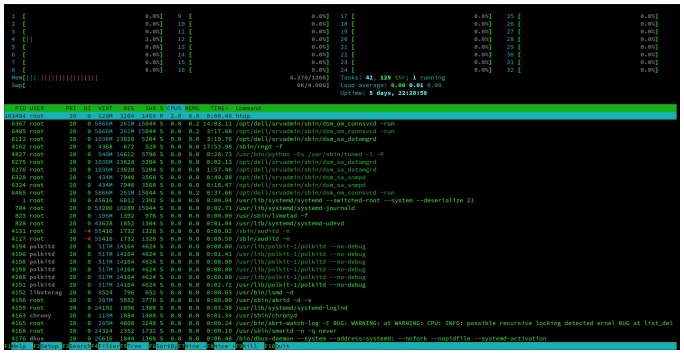
HV Benchmarking

In order to determine if there are any potential bottlecks in the Hypervisor configuration some profiling was done.

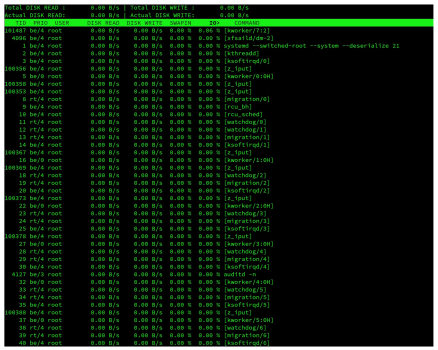
The main 4 areas which were profiled were:

- VM Configuration
- Networking
- Storage Configuration
- RAM Usage

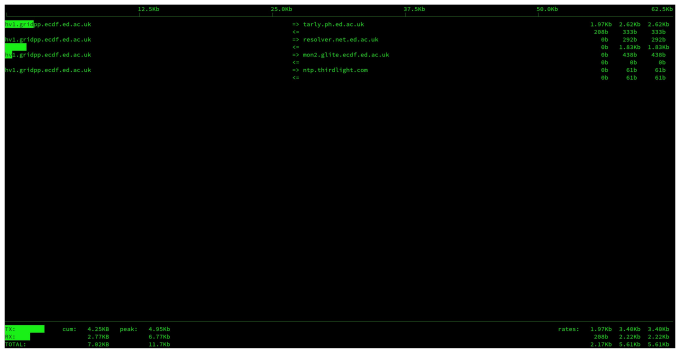
Performance Monitoring Tools used



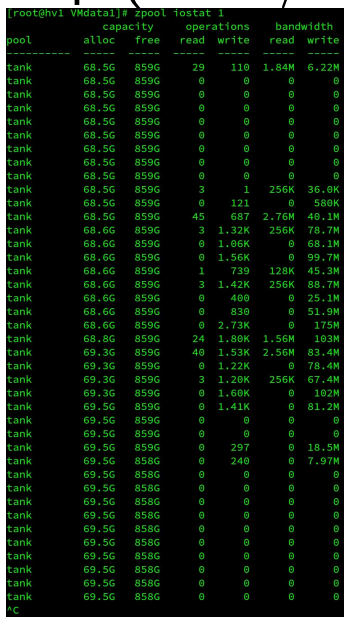
htop (performance)



iotop (disk-I/O)



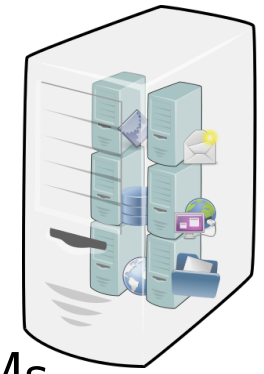
iftop (network)



```
[root@hv1 VMdata1]# zpool iostat 1
               capacity     operations     bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
tank        68.5G   859G     29    110  1.84M  6.22M
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      0      0      0      0
tank        68.5G   859G      3      1   256K  36.0K
tank        68.5G   859G      0    121      0   580K
tank        68.5G   859G     45    687  2.76M  48.1M
tank        68.6G   859G      3  1.32K   256K  78.7M
tank        68.6G   859G      0  1.06K      0  68.1M
tank        68.6G   859G      0  1.56K      0  99.7M
tank        68.6G   859G      1    739   128K  45.3M
tank        68.6G   859G      3  1.42K   256K  88.7M
tank        68.6G   859G      0    400      0  25.1M
tank        68.6G   859G      0    830      0  51.9M
tank        68.6G   859G      0  2.73K      0   175M
tank        68.8G   859G     24  1.80K  1.56M   103M
tank        69.3G   859G     40  1.53K  2.56M  83.4M
tank        69.3G   859G      0  1.22K      0  78.4M
tank        69.3G   859G      3  1.20K   256K  67.4M
tank        69.3G   859G      0  1.60K      0   102M
tank        69.5G   859G      0  1.41K      0  81.2M
tank        69.5G   859G      0      0      0      0
tank        69.5G   859G      0      0      0      0
tank        69.5G   859G      0    297      0  18.5M
tank        69.5G   858G      0    240      0  7.97M
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
tank        69.5G   858G      0      0      0      0
^C
```

zpool (ZFS-I/O)

Checking the VM Configuration



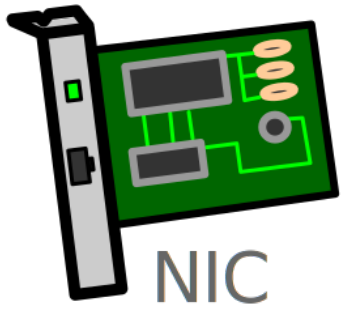
We use qemu+kvm and shell scripts to manage the VMs. This gives us a lot of control over the VM configuration.

Using paravirtualisation with `virtio` devices in the VM configs and using qcow2 disk images.

Running KVM VMs on DRBD is used in production elsewhere so is a proven technology.

Conclusion: VM configuration is not a bottleneck.

Profiling the Network Performance



Using a dedicated 10Gb link
between Hypervisors.
This is configured with static IPs and fixed bandwidth.

DRBD has to be configured to use all available bandwidth on dedicated links.

Monitoring the fiber link during heavy I/O ops shows high bandwidth usage ($\approx 1Gb$).

(Output from monitoring DRBD traffic throughput with `iftop`)

Conclusion: Network configuration is not a bottleneck.

Storage Profiling (1/5)

Testing the write performance showed a performance drop in using just ZFS+ZVol vs ZFS.

A configuration parameter which can effect the ZVol performance is the `volblocksize`.
(This controls the block size of the ZVol virtual block device).

Additionally the `recordsize` parameter for the zpool layer can impact the underlying ZFS performance.
(This is effectively the maximum block size available on the ZFS filesystem).

I have explored the impact of varying these on the Hypervisors performance.

Storage Profiling (2/5)

When benchmarking the performance of the filesystem we attempted to turn off all of the system caches.

Turn off ZFS read-cache (Adjustable Replacement Cache (ARC)):

```
zfs set primarycache=metadata tank
zfs set secondarycache=metadata tank
```

"Turn off" the VM cache for DRBD and EXT4:

```
echo 3 > /proc/sys/vm/drop_caches
echo 1 > /proc/sys/vm/dirty_ratio
echo 1 > /proc/sys/vm/dirty_background_ratio
```

All other ZFS settings have been left as default.

Storage Profiling (3/5)

To test the flush speed to disk:

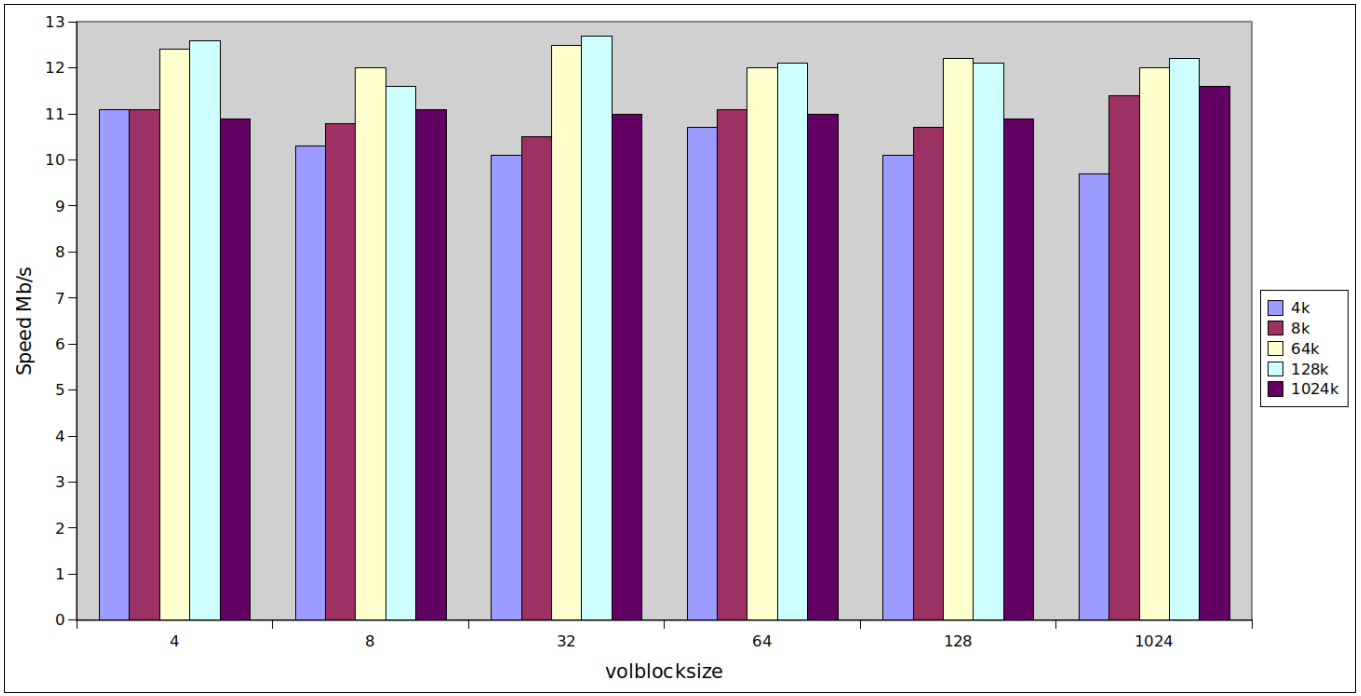
```
dd if=<(openssltry enc -aes-256-ctr -pass \
  pass:"$(dd if=/dev/urandom bs=128 count=1 \
  2>/dev/null  base64)" -nosalt < /dev/zero) \
  of=/VMdata1/testFile bs=1M count=1000 iflag=fullblock \
  oflag=sync
```

To try and run a (quick) representative benchmark of filesystem use:

```
fio --randrepeat=1 --ioengine=libaio --gtod_reduce=1 \
   --name=test --filename=/VMdata1/fio-test-128 --bs=4k \
   --iodepth=64 --size=1000M --readwrite=randrw --rwmixread=75
```

Storage Profiling (4/5)

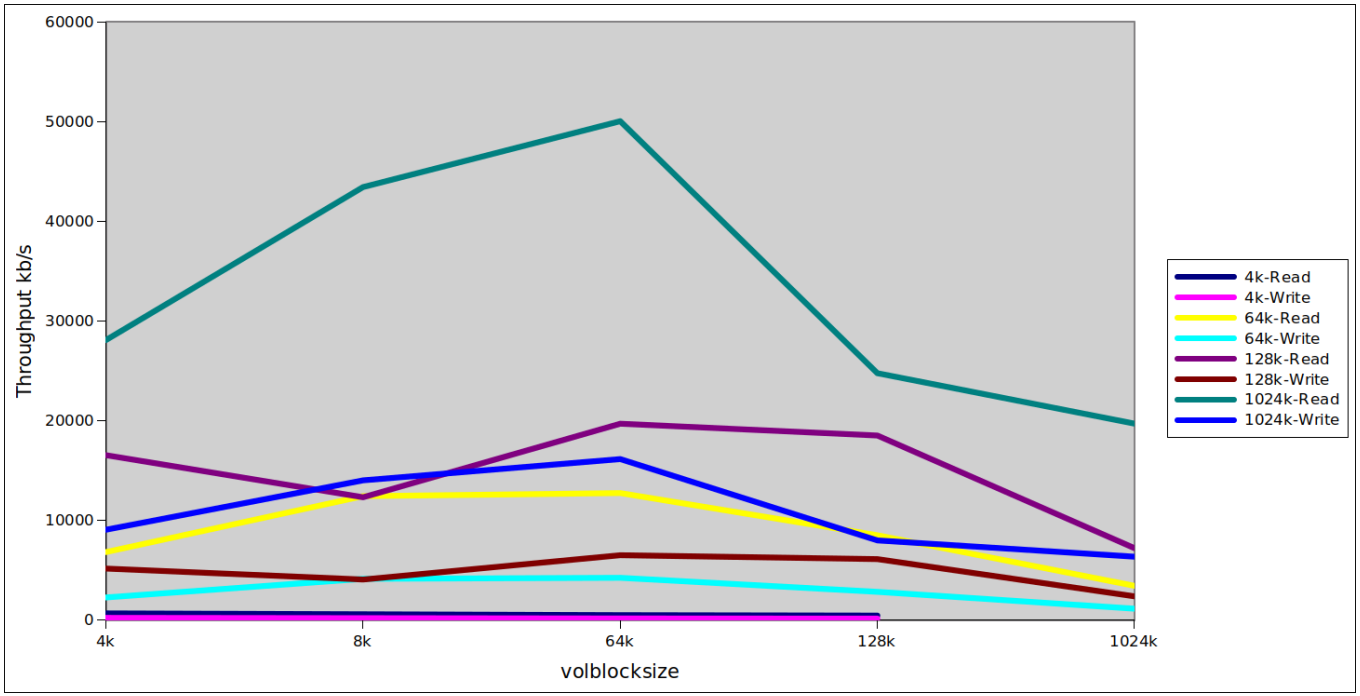
The results of varying both `volblocksize` and `recordsize` on the flush speed using `dd` are shown below:



For reference running this test on just ZFS gets 33Mb/s.

Storage Profiling (5/5)

The results of varying `volblocksize` and `bs` in the `fio` benchmarks are shown below:



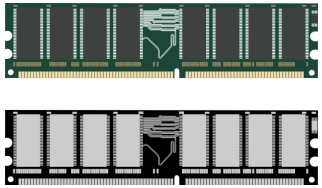
Storage Profiling (Conclusion)

Tuning the `volblocksize` of the ZVol layer indicates that the best value is either 64kb or 128kb.

Varying `recordsize` has no effect on speed at which data can be flushed to disk.

Conclusion: We changed the `volblocksize` of the ZVol to be 64kb.

Profiling RAM use on the Hypervisors

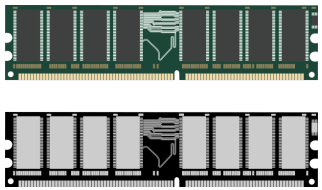


We observed that the full memory of the hypervisor, 128Gb, was consumed whilst just running a 60Gb VM.

In order to understand if this was effecting our read/write performance we decided to run some tests on another machine with ZFS storage.

Whilst testing on this machine we left the ZFS settings to the default values.

Profiling RAM use on the Hypervisors



The test machine had 16Gb of RAM.
Filled 9Gb with randomly generated data stored in tmpfs. Then wrote a 10Gb file to ZFS storage.

Once the ZFS (ARC) read-cache reached 7Gb in size the write performance on the machine began to suffer.
At this stage load on system drastically increased and write performance dropped.

This can be avoided by by tuning the ZFS cache or turning it off.

```
echo $MAXVALUE > /sys/module/zfs/parameters/zfs_arc_max
```

Conclusion: We need to limit the size of the ARC cache if we run low on RAM

Changes to isolate the Guest VMs from the Host (1/3)

Copying large file(s) ($\approx$ 10Gb) on the Host would cause guest VMs to lock up.
This is needed when deploying new VM images.
This is related to the flushing of the DRBD or ext4 write caches to disk.

Large writes would fill the write buffer with no disk activity.

A few seconds later the kernel would begin to flush the cache.

During the flush all of the device I/O is used for writing out data.

This caused all other I/O operations to the DRBD layer to pause until the flush was complete.

Changes to isolate the Guest VMs from the Host (2/3)

The best solution we found was to minimise the dirty caches within the kernel:

```
echo 1 > /proc/sys/vm/dirty_background_ratio
echo 1 > /proc/sys/vm/dirty_ratio
echo 1 > /proc/sys/vm/dirty_writeback_centisecs
```

This change reduces the dirty cache from 10% of the system memory (defult) to 1%. With a maximum time of 0.01sec that data can be buffered before being flushed.

This had the impact that although data is still preferentially be written to the write cache first there isn't a long pause whilst this is then flushed to disk.

Changes to isolate the Guest VMs from the Host (3/3)

Processes on the Host don't share access to the disk resources in a friendly way.

In order to enforce that no process consumes all of the available I/O we decided to use cgroups.
We found that limiting each individual process to 80% of the maximum disk bandwidth was effective.

```
systemctl edit user-0.slice
"[Slice]
BlockIOReadBandwidth=/dev/drbd1 83886080
BlockIOWriteBandwidth=/dev/drbd1 83886080"
```

These changes were verified by testing the block device with the command:

```
hdparm --direct -t /dev/drbd1
```

Review of Changes Made

- DRBD needed to be configured to use the maximum network bandwidth available even on a dedicated link
- ZFS ARC cache on the HV has been adjusted to use less memory
- DRBD+Ext4 write cache has been reduced
- `cgroup` configuration constrains individual processes to 80% of max disk throughput

Ongoing Work

Have now understood what the main impacts on the performance of the Hypervisor are.
Before we return our production middleware services to the hypervisor I plan to:

- Run a database benchmark from within a guest VM and check performance.
- Perform a full "acid test" to verify DRBD as recovery tool (power-off with live-VM).
- Run any additional performance tests.

Conclusions

- Have understood how to deploy and optimize DRBD.
- Can optimise ZFS for use as soft RAID mirror in hypervisors.
- Can deploy grid services in VMs with high redundancy.
- Suggestions welcome as work is ongoing.

Thanks for listening