Tips and Recommendations for Storage Server Tuning

Table of Contents (Page)

General Notes

This page presents some tips and recommendations on how to improve the performance of BeeGFS storage servers. As usual, the optimal settings depend on your particular hardware and usage scenarios, so you should use these settings only as a starting point for your tuning efforts. Benchmarking tools such as IOR, IOZone or StorageBench would help you identify the best settings for your BeeGFS storage servers.

Some of the settings suggested here are non-persistent and will be reverted after the next reboot. In order to keep them permanently, you could add the corresponding commands to `/etc/rc.local`, as seen in the example below, use `/etc/sysctl.conf` or create `udev` rules to reapply them automatically when the machine boots.

```
echo 5 > /proc/sys/vm/dirty_background_ratio
echo 20 > /proc/sys/vm/dirty_ratio
echo 50 > /proc/sys/vm/vfs_cache_pressure
echo 262144 > /proc/sys/vm/min_free_kbytes
echo 1 > /proc/sys/vm/zone_reclaim_mode

echo always > /sys/kernel/mm/transparent_hugepage/enabled
echo always > /sys/kernel/mm/transparent_hugepage/defrag

devices=(sda sdb)
for dev in "${devices[@]}"
do
  echo deadline > /sys/block/${dev}/queue/scheduler
  echo 4096 > /sys/block/${dev}/queue/nr_requests
  echo 4096 > /sys/block/${dev}/queue/read_ahead_kb
```

```
  echo 256 > /sys/block/${dev}/queue/max_sectors_kb
done
```

Regardless of which type of RAID your system is using (either hardware or software based), there are some general guidelines that should be considered.

- Mixing devices of different models and manufacturers inside the same RAID volume can have a negative impact on performance, as the slowest device in a volume will define the maximum speed of the volume.

- Small disk arrays (around 6 disks) are recommended for workloads characterized by write operations on small amounts of data. The less disks in the array, the shorter the array data stripe and the less data read for parity updates on every write.

- Large disk arrays decrease the amount of space occupied by parity bits and increase the net storage capacity of the RAID volume. This increases the throughput of the device. 10 to 12 disks per RAID6 volume is usually a good balance between capacity, fault tolerance and performance.

- Smaller disks (e.g. 2 or 4 TB) in the array allow for faster array rebuilds.

- Avoiding architectural bottlenecks (e.g. using multiple RAID controllers, high-speed networks and PCI interfaces) usually is more important for boosting performance of the system than any tuning option of block devices, underlying file system, or operating system.

Hardware RAID

Partition Alignment & RAID Settings of Local File System

To get the maximum performance out of your storage devices, it is important to set each partition offset according to their respective native alignment. Check the page Partition Alignment Guide for a walk-through about partition alignment and creation of a RAID-optimized local file system.

A very simple and commonly used method to achieve alignment without the challenges of partition alignment is to completely avoid partitioning and instead, create the file system directly on the device, as shown in the following sections.

Storage Server Throughput Tuning

In general, the BeeGFS storage service can use any standard Linux file systems. However, XFS is the recommended file system for disk partitions of storage targets, because it scales very well for RAID arrays and typically delivers a higher sustained write throughput on fast storage, compared to alternative file systems. There have been significant improvements of ext4 streaming performance in recent Linux kernel versions, but XFS is still the best choice.

The default Linux kernel settings are rather optimized for single disk scenarios with low IO concurrency, so there are various settings that need to be tuned to get the maximum performance out of your storage servers.

Formatting Options

Make sure to enable RAID optimizations of the underlying file system, as described in the last section of the page: Create RAID-optimized File System.

While BeeGFS uses dedicated metadata services to manage global metadata, the metadata performance of the underlying file system on storage servers still matters for operations like file creates, deletes, small reads/writes, etc. Recent versions of XFS (similar work in progress for ext4) allow inlining of data into inodes to avoid the need for additional blocks and the corresponding expensive extra disk seeks for directories. In order to use this efficiently, the inode size should be increased to 512 bytes or larger.

The example below shows the command for creating an XFS partition with larger inodes on 8 disks (where the number 8 does not include the number of RAID-5 or RAID-6 parity disks) and 128 KB chunk size.

```
$ mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k -isize=512 /dev/sdX
```

Mount Options

Enabling last file access time is inefficient, because it means that the file system needs to update the timestamp by writing data to the disk even in cases when the user only reads file contents or when the file contents have already been cached in memory and no disk access would have been necessary at all. (Note: Recent Linux kernels have switched to a new "relative atime" mode, so setting `noatime` might not be necessary in these cases.)
If your users don't need last access times, you should disable them by adding "`noatime`" to your mount options.

Increasing the number of log buffers and their size by adding `logbufs` and `logbsize` mount options allows XFS to generally handle and enqueue pending file and directory operations more efficiently.

There are also several mount options for XFS that are intended to further optimize streaming performance on RAID storage, such as `largeio`, `inode64`, and `swalloc`.

If you are using XFS and want to go for optimal streaming write throughput, you might also want to add the mount option `allocsize=131072k` to reduce the risk of fragmentation for large files. Please, consider that this setting could have a significant impact on the interim space usage in systems with many parallel write and create operations.

If your RAID controller has a battery-backup-unit (BBU) or similar technology to protect the cache contents on power loss, adding the mount option `nobarrier` for XFS or ext4 can significantly increase throughput. Make sure to disable the individual internal caches of the attached disks in the controller settings, as these are not protected by the RAID controller battery.

The example below shows the command for mounting an XFS partition of a RAID volume on a server with a RAID controller battery.

```
$ mount -onoatime,nodiratime,logbufs=8,logbsize=256k,largeio,inode64,swalloc,allocsize=131072k,nobarri
```

IO Scheduler

First, set an appropriate IO scheduler for file servers:

```
$ echo deadline > /sys/block/sdX/queue/scheduler
```

Now give the IO scheduler more flexibility by increasing the number of schedulable requests:

```
$ echo 4096 > /sys/block/sdX/queue/nr_requests
```

To improve throughput for sequential reads, increase the maximum amount of read-ahead data. The actual amount of read-ahead is adaptive, so using a high value here won't harm performance for small random access.

```
$ echo 4096 > /sys/block/sdX/queue/read_ahead_kb
```

Virtual Memory Settings

To avoid long IO stalls (latencies) for write cache flushing in a production environment with very different workloads, you will typically want to limit the kernel dirty (write) cache size:

```
$ echo 5 > /proc/sys/vm/dirty_background_ratio
$ echo 10 > /proc/sys/vm/dirty_ratio
```

In special use-cases, if you are going for optimal sustained streaming performance, you may instead want to use different settings that start asynchronous writes of data very early and allow the major part of the RAM to be used for write caching. (For generic use-cases, use the settings described above, instead.)

```
$ echo 1 > /proc/sys/vm/dirty_background_ratio
$ echo 75 > /proc/sys/vm/dirty_ratio
```

Assigning slightly higher priority to inode caching helps to avoid disk seeks for inode loading:

```
$ echo 50 > /proc/sys/vm/vfs_cache_pressure
```

Buffering of file system data requires frequent memory allocation. Raising the amount of reserved kernel memory will enable faster and more reliable memory allocation in critical situations. Raise the corresponding value to 64 MB if you have less than 8GB of memory, otherwise raise it to at least 256 MB:

```
$ echo 262144 > /proc/sys/vm/min_free_kbytes
```

Transparent huge pages can cause performance degradation under high load, due to the frequent change of file system cache memory areas. For RHEL 5.x, RHEL 6.x and derivatives, it is recommended to disable default transparent huge pages support, unless huge pages are explicity requested by an application through madvise:

```
$ echo madvise > /sys/kernel/mm/redhat_transparent_hugepage/enabled
$ echo madvise > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

For RHEL 7.x and other distributions, it is recommended to have transparent huge pages enabled:

```
$ echo always > /sys/kernel/mm/transparent_hugepage/enabled
$ echo always > /sys/kernel/mm/transparent_hugepage/defrag
```

Controller Settings

Optimal performance for hardware RAID systems often depends on large IOs being sent to the device in a single large operation. Please refer to your hardware storage vendor for the corresponding optimal size of `/sys/block/sdX/max_sectors_kb`. It is typically good if this size can be increased to at least match your RAID stripe set size (i.e. chunk_size x number_of_disks):

```
$ echo 1024 > /sys/block/sdX/queue/max_sectors_kb
```

Furthermore, high values of sg_tablesize (`/sys/class/scsi_host/.../sg_tablesize`) are recommended to allow large IOs. Those values depend on controller firmware versions, kernel versions and driver settings.

ZFS (Software RAID)

Software RAID implementations demand more powerful machines than traditional systems with RAID controllers, especially if features like data compression and checksums are enabled. Therefore, using ZFS as the underlying file system of storage targets will require more CPU power and RAM than a more traditional BeeGFS installation with hardware RAID. It will also increase the importance of disabling features like CPU frequency scaling.

It is also recommended to be economical with the options enabled in ZFS, e.g. a feature like deduplication uses a lot of resources and can have a significant impact on performance. Also keep in mind that a feature like compression might spoil the result of a streaming benchmark like `IOR`.

Another important factor that impacts performance in such systems is the version of ZFS packages used. At the time of writing, the latest version 0.7.1 still had some performance issues. In our internal tests, the highest throughput was observed with version 0.6.5.11.

Setting ZFS Module Parameters

After loading the ZFS module, please set the module parameters below, before creating any ZFS storage pool.

Maximum Record Size

Raise the maximum size of data blocks that can later be defined for each ZFS storage pool. This can be done by setting a new value to parameter `zfs_max_recordsize`, in bytes, as follows. Depending on the workload on your system, you may want to set different record sizes for the ZFS pools, so raising this limit will give you more tuning possibilities later.

```
$ echo 4194304 > /sys/module/zfs/parameters/zfs_max_recordsize
```

IO Scheduler

Set the IO scheduler used by ZFS. Both `noop` and `deadline`, which implement simple scheduling algorithms, are good options, as the storage daemon is run by a single Linux user.

```
$ echo deadline > /sys/module/zfs/parameters/zfs_vdev_scheduler
```

Read Chunk Size

Data is read by ZFS in data chunks of a certain size. Increasing such size could speed up read operations.

```
$ echo 1310720 > /sys/module/zfs/parameters/zfs_read_chunk_size
```

Data Prefetch

When processing a read request, ZFS can predict which extra data might be requested by future read requests, and use the opportunity to read that data in advance, so that when it is needed, it is already found in memory. This option is disabled by default in some ZFS releases. Set this option to `0` to activate data prefetching if you are using spinning disks. Set it to `1` to disable it if you are using flash devices like SSDs.

```
$ echo 0 > /sys/module/zfs/parameters/zfs_prefetch_disable
```

Data Aggregation Limit

ZFS is able to aggregate small IO operations that handle neighboring or overlapping data into larger operations, in order to reduce the number of IOPs. The option `zfs_vdev_aggregation_limit` sets the maximum amount of data that can be aggregated, before the IO operation is finally performed on the disk.

```
$ echo 262144 > /sys/module/zfs/parameters/zfs_vdev_aggregation_limit
```

Creating ZFS Pools for Storage Targets

Basic options like the pool type, cache and log devices must be defined at the creation time of the pool, as seen in the example below. The mount point is optional, but it is a good practice to define it with option `-m`, in order to control where the storage target directory will be located.

```
$ zpool create -m /data/storage001 storage001 raidz2 sda sdb sdc sdd sde sdf sdg sdh sdi sdj sdk sdl
```

Data Protection

RAID-Z2 is the recommended pool type, as it offers a good balance between data safety, storage space overhead and performance overhead in comparison with single-parity (mirrored) and triple-parity pools. The ideal number of devices in each pool depends on the IO patterns that are more common in the system. In most cases, 12 drives was the pool size that produced the best performance results.

Partition Alignment

The list of devices composing the storage pool should contain whole block devices, so that ZFS can create disk partitions with optimal alignment. Moreover, including partitions formatted with other file systems like XFS and ext4 to this list introduces an unnecessary extra software layer to the system with a significant performance overhead.

Log Devices

In order to provide applications with an extra level of data integrity, you should consider enabling ZIL, the ZFS Intent Log. Similarly to a database log, ZIL keeps record of write operations performed on the pool, allowing them to be committed or rolled back, in the case of a system crash.

ZIL is enabled by adding log devices to a pool. For the sake of performance, flash devices should be used for that purpose. The command below shows an example of how log devices can be added to a pool.

```
$ zpool add storage001 log mirror sdx sdy
```

By default, the ZIL handles only synchronous (not cached) write operations. Therefore, as the BeeGFS storage service mostly performs asynchronous (cached) write operations, ZIL with the out-of-the-box settings will be used by BeeGFS only when applications call `fsync()`, similar to the behavior of applications directly accessing a ZFS file system. As most applications rarely make use of this `fsync()` syscall, the ZIL is usally rather small in comparison to the flash read cache.

The command below can change this behavior by making all write operations go through the ZIL. However, this is likely to cause performance degradation in many cases, e.g. when the streaming speed of the ZIL is lower than the streaming speed of the backend RAID array, so this setting is not recommended.

```
$ zfs sync=always
```

Since the ZIL is usually rather small, one typical choice is to use only a minor partition of a flash device as ZIL and use the major part of the flash device as read cache.

Cache Devices

Adding flash devices as read cache can be beneficial for repeated read operations of the same data, depending on the workload and speed of the flash devices in comparison to the backend disk array. The command below shows an example of how cache devices can be added to a pool.

```
$ zpool add storage001 cache sdz
```

Block Size

ZFS queries block devices to find out their sector size, which will be used as the block size of the `vdevs` that compose the storage pool. Unfortunately, some block devices report the wrong sector size, causing the storage pool to end up using the wrong block size. In order to fix that, you should set the ZFS property `ashift=9` if you have 512 byte sector disks or `ashift=12` if you have 4 KB sector disks. (Use `ashift=12` if you are not sure.)

```
$ zpool create -o ashift=12 -m /data/storage001 storage001 raidz2 sda sdb sdc sdd sde sdf sdg sdh sdi
```

Configuring ZFS Pools for Storage Targets

File system properties like data compression and record size, may be defined at creation time with option `-O`, as shown in the example below, but they may also be defined (or redefined) later with command `zfs set`, as seen in the following sections.

```
$ zpool create -f -O compression=off -O recordsize=4M -o ashift=12 -O atime=off -O xattr=off
-m /data/storage001 storage001 raidz2 sda sdb sdc sdd sde sdf sdg sdh sdi sdj sdk sdl log sdm
```

Data Compression

Data compression is a feature that should enabled only if the data of the files stored in the pool allows a high degree of compression. Otherwise, the CPU overhead caused by the compression functions will not be compensated by the decrease of the amount of data involved in the IO operations.

```
$ zfs set compression=off storage001
```

If you enable data compression, please use the data compression algorithm lz4, which is known to have a good balance between compression ratio and performance.

```
$ zfs set compression=lz4 storage001
```

Record Size

The record size is the unit that ZFS validates by calculating checksums. Larger data blocks can reduce the frequency in which checksums are calculated, and speed up write operations. On the other hand, such data blocks are read every time a write operation is performed on them. In scenarios where the block is much larger than the amount of data handled by write operations, this could have a negative performance impact.

```
$ zfs set recordsize=4m storage001
```

Deduplication

Depuplication (dedup) is a space-saving feature that works by keeping a single copy of multiple identical files stored in the ZFS file system. This feature has a significant performance impact and should be disabled if the system has plenty of storage space.

```
$ zfs set dedup=off storage001
```

Unnecessary Properties

The BeeGFS storage service does not update access time and does not use extended attributes. So, these properties may be disabled in ZFS pools, as follows.

```
$ zfs set atime=off storage001
$ zfs set xattr=off storage001
```

System BIOS & Power Saving

To allow the Linux kernel to correctly detect the system properties and enable corresponding optimizations (e.g. for NUMA systems), it is very important to keep your system BIOS updated.

The dynamic CPU clock frequency scaling feature for power saving, which is typically enabled by default, has a high impact on latency. Thus, it is recommended to turn off dynamic CPU frequency scaling. Ideally, this is done in the machine BIOS, where you will often find a general setting like "Optimize for performance".

If frequency scaling is not disabled in the machine BIOS, recent Intel CPU generations require the parameter "`intel_pstate=disable`" to be added to the kernel boot command line, which is typically defined in the `grub` boot loader configuration file. After changing this setting, the machine needs to be rebooted.

If the Intel pstate driver is disabled or not applicable to a system, frequency scaling can be changed at runtime, e.g. via:

```
$ echo performance | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor >/dev/null
```

You can check if CPU frequency scaling is disabled by using the following command on an idle system. If it is disabled, you should see the full clock frequency in the "`CPU MHz`" line for all CPU cores and not a reduced value.

```
$ cat /proc/cpuinfo
```

Concurrency Tuning

Worker Threads

Storage servers, metadata servers and clients allow you to control the number of worker threads by setting the value of `tuneNumWorkers` (in `/etc/beegfs/beegfs-X.conf`). In general, a higher number of workers allows for more parallelism (e.g. a server will work on more client requests in parallel). But a higher number of workers also results in more concurrent disk access, so especially on the storage servers, the ideal number of workers may depend on the number of disks that you are using.

Back to User Guide - Tuning and Advanced Configuration

[Source] 2018-02-08 22:30:28

Valid XHTML :: Valid CSS: :: Powered by WikkaWiki