Gerrit Performance Cheat Sheet

FACTORS | DEFINITION

Number of Users

The number of users is only an indirect factor for Gerrit tuning as most Git operations are done completely offline.
The more users you have, the more repositories and push/fetch requests you will probably encounter.
The majority of load is typically caused by build systems (CI).
The biggest enterprise instance we have seen has 15k active users.

Number of Repositories

The number of repositories (Gerrit projects) determines how much disk space you need.
We have seen instances with more than 10k repositories but would not recommend more than 2500 per server.

Number of Fetch Requests

This is probably the most important tuning factor.
To improve throughput, fetch requests should be handled in parallel, but parallel cloning needs CPUs as well as memory.
A Gerrit server optimized for heavy load (32 cores, 32 GB RAM) can handle about 1M fetch requests per day, processing up to 50 in parallel.

Related gerrit.config Options

- sshd.threads
- sshd.batchThreads
- sshd.commandStartThreads
- httpd.maxThreads
- container.heapLimit
- database.poolLimit
- database.poolMaxIdle

Number of Push Requests

In most enterprise settings, push requests contribute less than one percent to the number of total operations.
Because of this, their number can be typically neglected.

Related gerrit.config Options

- receive.timeout
- core.packedGitLimit
- core.packedGitWindowSize
- core.packedGitOpenFiles
- gc.interval

Protocol for Fetch/Push

SSHD **HTTP**

Ssh connections consume about 0.9 CPUs in avarage during a fetch/push operation. Http connections consume about 0.5 CPUs, https adds additional encryption traffic and makes it comparable to ssh. Ssh is recommended for CI users as this allows push based notifications (see CI info box).

Repository Size

Repository size determines the amount of storage you need on disk. In addition, it influences the needed memory during a clone request as pack files have to be loaded and streamed. The largest repository on disk should still fit in 1/4 of your heap. Garbage collection across all projects will take longer, the more repository data has to be processed. Gerrit can handle at least 1TB of total repository data easily.

Gerrit Performance Cheat Sheet

HARDWARE SIZING

Nunber of Cores

For every CPU core you add, you can handle up to 2 parallel fetch/push requests. Our experience tells 32 cores per 1M daily fetch requests is pretty common. Scale this number up/down based on your load.

RAM

You should have at least <#Cores> GB heap allocated for Gerrit, the more memory, the better.
Our experience tells 32 GB per 1M daily requests is pretty common.

Storage

Storage needs are determined by the Git repository sizes. Fast storage (SSDs or NAS) really pays off as git fetch, push and gc are all IO heavy.

Network

The higher the network bandwidth, the shorter it will take to fetch and push repositories.
Depending on the avarage Git repository size and number of parallel requests, network connectivity can become the primary bottleneck.
Most enterprises have Gigabit connections.

Number of Servers

Whenever horizontal scaling is not cost efficient any more, we recommend setting up another server.
If the number of repositories exceeds 2500, a new server should be used as well or reviews will get painfully slow.
Use Gerrit's replication feature to synch repo content and permissions to servers in different geographies if network is the limiting factor.

Gerrit Performance Cheat Sheet

TUNING RELATED GERRIT.CONFIG OPTIONS

S Small Instance (<100k requests per day, 4 GB RAM, 4 Cores)
M Medium Instance (around 500k requests per day, 16 GB RAM, 16 Cores)
L Large Instance (around 1M requests per day, 32 GB RAM, 32 Cores)

sshd.threads:

S 8
M 32
L 64

#threads to process ssh requests, limiting the number of possible parallel clones/pushes.
Ssh connections consume about 0.9 CPUs per parallel fetch/push operation.
Defaults to 1.5 • <#Cores>, we rather recommend 2 • <#Cores>
If you run into heap space issues, scale down this number again.

database.poolLimit:

S 50
M 150
L 250

#DB connections for Gerrit.
As a fetch/push request or a review action can consume multiple connections, set at least to <sshd.threads> + <httpd.maxThreads>

receive.timeout:

S 4 min
M 4 min
L 4 min

Timeout to process incoming changes and update refs and Gerrit changes.
Default of 2min is typically too small for huge repositories.

container.heapLimit:

S 4g
M 16g
L 32g

Java heap used for Gerrit. The more repository data Gerrit can cache in memory, the better. You should have at least <#Cores> GB size heap size allocated for Gerrit. The largest repository on disk should still fit in 1/4 of your heap. 32 GB per 1M daily requests is pretty common.

httpd.maxThreads:

S 25
M 50
L 100

#threads to process http clone/push requests and review related activities.
Http consume about 0.5 cores, https adds additional encryption traffic and makes it comparable to ssh.

sshd.batchThreads:

S 2
M 4
L 8

#threads reserved to users in a Gerrit group with the BATCH capability. This allows to separate CI users causing heavy load from human users by placing their requests in different thread pools.
Interactive users will have <sshd.threads> - <sshd.batchThreads> available just for them.
This can improve clone/push performance for human users significantly.

core.packedGitLimit:

S 1g
M 4g
L 8g

Maximum cache size to store Git pack files in memory. The default (10 MB) is way too small if you frequently clone large repositories and like to cache their data.
1/4 of your heap size is a common choice.

database.poolMaxIdle:

S 16
M 16
L 16

Maximum time before a DB connections gets released.
As DB pool size is typically increased from its default value, this parameter should be too.

sshd.CommandStartThreads:

S 2
M 3
L 5

#threads used to process incoming ssh connection requests.
Setting should only be adjusted for CI systems that create a burst of connection requests in parallel.
Especially in AOSP build environments increasing this value helps reducing the avarage wait queue size.

gc.interval:

S 1 week
M 3 day
L 1 day

Determines how often Gerrit garbage collection (JGit gc) is run across all repositories. Running JGit gc frequently is crucial for good fetch/push performance as well as a smooth source code browsing experience. JGit gc is more efficient than command line git garbage collection and causes no problems with Gerrit running in parallel. Parameters to control JGit gc's resource consumption are in ~gerrit/.gitconfig. Don't forget to set gc.startTime for the initial garbage collection time.

core.packedGitWindowSize:

S 8k
M 16k
L 16k

Number of bytes of a pack file to load into memory in a single read operation.
16k is a common choice.

core.packedGitOpenFiles:

S 1024
M 2048
L 4096

Maximum number of pack files to have open at once. If you increased packedGitLimit, you have to adjust this value too. If you increase this to a larger setting you may need to also adjust the ulimit on file descriptors for the host JVM, as Gerrit needs additional file descriptors available for network sockets and other repository data manipulation.

Gerrit Performance Cheat Sheet

GARBAGE COLLECTION (~GERRIT/.GITCONFIG)

S	Small Instance (<100k requests per day, 4 GB RAM, 4 Cores)
M	Medium Instance (around 500k requests per day, 16 GB RAM, 16 Cores)
L	Large Instance (around 1M requests per day, 32 GB RAM, 32 Cores)

pack.threads:

S	1
M	4
L	8

#threads used for Gerrit (JGit) garbage collection. 1/4 • <#Cores> is a common choice.

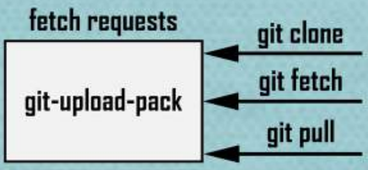
pack.WindowMemory:

S	1g
M	4g
L	8g

Use this setting to control how much memory (Java heap) is used for Gerrit garbage collection (JGit gc). 1/4 of the configured Java heap is a common choice.

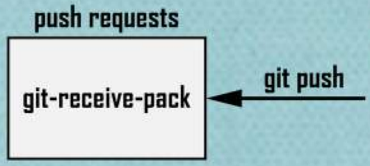
GERRIT PERFORMANCE TUNING FAQ

What are fetch/push requests and how many will I have per day?



A **fetch request** is peformed for git commands **clone, fetch** and **pull**.
It updates the local repository with changes that happened on the server since the last fetch.
Fetch requests typically contribute more than 99 percent to the server load.
A **push request** is performed by git push and updates the remote server with the local changes.

Gerrit logs all ssh fetch and push requests in its **~gerrit/logs/sshd_log** file including time stamp, session, user, command, repository, processing time, queue time and exit code (in that order).
http request logging has to be explicitly turned on (**httpd.requestLog=true**) and will then show up in **~gerrit/logs/httpd_log**

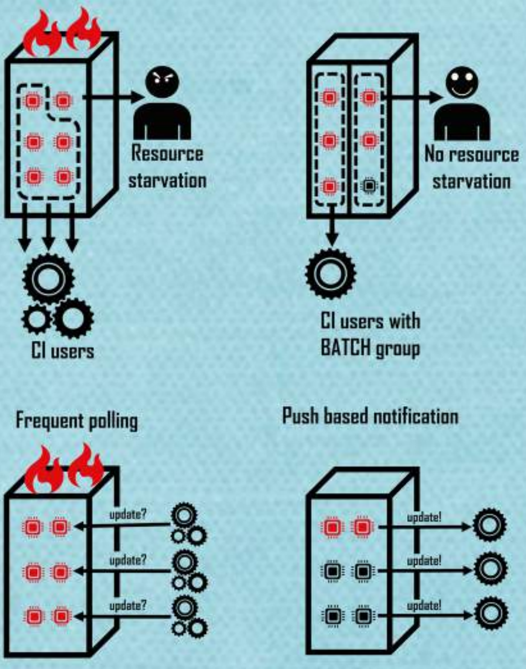


Fetch requests contain the term **git-upload-pack**, push requests the term **git-receive-pack**.
Basic Unix commands should be sufficient to analyze how many requests your server gets per day.
For instance, to count the number of ssh fetch requests, use

```
fgrep "git-upload-pack" sshd_log | wc -l
```

For more advanced analysis we recommend tools like Splunk, ELK or Graylog.
The Gerrit command **show-connections** gives you an idea how many parallel ssh requests are waiting to be processed, it should not be more than 2 • <#Cores> in avarage.

How to deal with build users / heavy CI load?



Continuous Integration (**CI**) systems like Jenkins, Cruise Control or Team City have to frequently check for code changes to trigger their build jobs.
If an enterprise has hundreds of repositories and build jobs, CI users easily cause more than 90 percent of a Gerrit server's load.

Requests from interactive, human users may starve because of server overload. The first counter measurement is to assign all CI users to a Gerrit group which has the **BATCH capability** set. Configure **sshd.batchThreads** approriately to restrict those users to a subset of your total thread pool (and server resources).
Don"t fotget to update the BATCH group whenever a new CI user is added.

Gerrit also supports **push based notifications** which is way more resource efficient than the typical poll based approach. Against common belief push based notifications are also issued for ordinary branch updates and not just Gerrit changes.
The **Jenkins Gerrit Trigger Plugin** makes use of this feature and should be used as build job trigger whenever possible. Other CI systems have similar plugins that make use of Gerrit's push-based, **stream-events** command.
sshd.streamThreads can be used to fine tune performance for push based notifications but typically does not have to be modified.