Minecraft

- Optimization with Aikar's flags
- SystemD
- Atlauncher
- mcaselector on Linux

Optimization with Aikar's flags

Making a copy of this information.

Reference sources:

- aikar.co:tuning-the-jvm-g1gc-garbage-collector-flags-for-minecraft
- PaperMC:aikars-flags
- Pufferfish.host:how-to-apply-aikars-flags

TL;DR

Use these flags exactly, only changing Xmx and Xms. These flags work and scale accordingly to any size of memory, even 500MB but 1.15 will not do well with such low memory…)

> java -Xms10G -Xmx10G -XX:+UseG1GC -XX:+ParallelRefProcEnabled -XX:MaxGCPauseMillis=200 -XX:+UnlockExperimentalVMOptions -XX:+DisableExplicitGC -XX:+AlwaysPreTouch -XX:G1NewSizePercent=30 -XX:G1MaxNewSizePercent=40 -XX:G1HeapRegionSize=8M -XX:G1ReservePercent=20 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=4 -XX:InitiatingHeapOccupancyPercent=15 -XX:G1MixedGCLiveThresholdPercent=90 -XX:G1RSetUpdatingPauseTimePercent=5 -XX:SurvivorRatio=32 -XX:+PerfDisableSharedMem -XX:MaxTenuringThreshold=1 -Dusing.aikars.flags=https://mcflags.emc.gs -Daikars.new.flags=true -jar paperclip.jar nogui

LAST UPDATED: **Apr 25 2020 3:30PM EST**

Aikar's original post

Introduction

After many weeks of studying the JVM, Flags, and testing various combinations, I came up with a highly tuned set of Garbage Collection flags for Minecraft. I tested these on my server, and have been used for years. I then announced my research to the public, and to this day, many servers have been using my flag recommendations for years and reporting great improvement to garbage collection behavior.

These flags are the result of a ton of effort, and results of seeing it in production on various server sizes, plugin lists and server types. They have proven themselves repeatedly.

I strongly suggest using these flags to start your server. These flags help keep your server running CONSISTENT without any large garbage collection spikes. CPU may be slightly higher, but your server will be overall more reliable and stable TPS.

If these flags help your server, consider donating!

The JVM Startup Flags to use – MC 1.15 (Java 8+, MC 1.8+) Update

Use these flags exactly, only changing Xmx and Xms. These flags work and scale accordingly to any size of memory, even 500MB but 1.15 will not do well with such low memory…)

> java -Xms10G -Xmx10G -XX:+UseG1GC -XX:+ParallelRefProcEnabled -XX:MaxGCPauseMillis=200 -XX:+UnlockExperimentalVMOptions -XX:+DisableExplicitGC -XX:+AlwaysPreTouch -XX:G1NewSizePercent=30 -XX:G1MaxNewSizePercent=40 -XX:G1HeapRegionSize=8M -XX:G1ReservePercent=20 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=4 -XX:InitiatingHeapOccupancyPercent=15 -XX:G1MixedGCLiveThresholdPercent=90 -XX:G1RSetUpdatingPauseTimePercent=5 -XX:SurvivorRatio=32 -XX:+PerfDisableSharedMem -XX:MaxTenuringThreshold=1 -Dusing.aikars.flags=https://mcflags.emc.gs -Daikars.new.flags=true -jar paperclip.jar nogui

LAST UPDATED: **Apr 25 2020 3:30PM EST**

These flags are recommended for ALL versions of Minecraft! 1.8 all the way to 1.15+, use this set.

IMPORTANT – READ – Don’t use ALL of your memory!! PTERODACTYL USERS!

When setting the Xms and Xmx values, if your host says you have 8000M memory, DO NOT USE 8000M! Minecraft (and Java) needs additional memory on top of that Xmx parameter. It is recommended to reduce your Xmx/Xms by about **1000-1500M** to avoid running out of memory or “OOMKiller” hitting your server. This also leaves room for the Operating System to use memory too.

Have 8000M memory? Use 6500M for safety. But you may also ask your host if they will cover this overhead for you and give you 9500M instead. Some hosts will! Just ask.

Recommended Memory

I recommend using at least 6-10GB, No matter how few players! If you can’t afford 10GB of memory, give as much as you can, but ensure you leave the operating system some memory too. G1GC operates better with more memory.

If you are running with 12GB or less memory for MC, you should not adjust these parameters.

If you are using an Xmx value greater than 12G

If you have and use more than 12GB of memory, adjust the following:

- -XX:G1NewSizePercent=40
- -XX:G1MaxNewSizePercent=50
- -XX:G1HeapRegionSize=16M
- -XX:G1ReservePercent=15
- -XX:InitiatingHeapOccupancyPercent=20

NOTICE: If you see increase in old generation collections after this, revert back to the base flags!

Explanation of these changes:

- Base flag set aims for 30/40 to reduce risk of to space issues. With more memory, less of an issue. We can give more to new generation with 40/50, as well as reduce reserve percent since the default reserve will already be larger.
- Region Size increase helps reduce humongous allocations, and speeds up remarking. We need a smaller region size at smaller heaps to ensure an adequate amount of regions available
- We can start looking for old generation memory to reclaim with more of a delay with IHOP at 20 since we have more old generation available to space on CPU.

Java GC Logging

Are you having old gen issues with these flags? Help me help you! Add the following flags based on your java version to enable GC Logging:

Java 8-10:

> -Xloggc:gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=1M

Java 11+:

> -Xlog:gc*:logs/gc.log:time,uptime:filecount=5,filesize=1M

Once you start seeing old generation collections in Timings, grab the logs/gc.log file (same location as your latest.log) and send it to me on Paper Discord to analyze.

GC logging does not hurt your performance and can be left on at all times. The files will not take up much space (5MB)

Technical Explanation of the Flags:

1. **-Xms matching -Xmx – Why:** You should never run your server with the case that -Xmx can run the system completely out of memory. Your server should always be expected to use the entire -Xmx!
   You should then ensure the OS has extra memory on top of that Xmx for non MC/OS level things. Therefore, you should never run MC with -Xmx settings you can’t support if java uses it all.Now, that means if -Xms is lower than -Xmx -YOU HAVE UNUSED MEMORY!

Unused memory is wasted memory.G1 (and probably even CMS to a certain threshold, but I'm only stating what I'm sure about) operates better with the more memory it's given. G1 adaptively chooses how much memory to give to each region to optimize pause time. If you have more memory than it needs to reach an optimal pause time, G1 will simply push that extra into the old generation and it will not hurt you (This may not be the case for CMS, but is the case for G1).The fundamental idea of improving GC behavior is to ensure short lived objects die young and never get promoted. With the more memory G1 has, the better assurance you will get that objects are not getting prematurely promoted to the old generation.G1 Operates differently than previous collectors and is able to handle larger heaps more efficiently.

If it does not need the memory given to it, it will not use it. The entire engine operates differently and does not suffer from too large of heaps, and this is industry wide accepted information that under G1 to keep Xms and Xmx the same!

2. **UnlockExperimentalVMOptions** – needed for some the below options
3. **G1NewSizePercent:** These are the important ones. In CMS and other Generations, tweaking the New Generation results in FIXED SIZE New Gen and usually is done through explicit size setting with -Xmn.With G1, things are better! You now can specify percentages of an overall desired range for the new generation. With these settings, we tell G1 to not use its default 5% for new gen, and instead give it 40%!**Minecraft has an extremely high a memory allocation rate, ranging to at least 800 Megabytes a second on a 30 player server! And this is mostly short lived objects (Block Position)**
   Now, this means MC REALLY needs more focus on New Generation to be able to even support this allocation rate. If your new gen is too small, you will be running new gen collections 1-2+ times per second, which is really bad.You will have so many pauses that TPS has risk of suffering, and the server will not be able to keep up with the cost of GC's.Then combine the fact that objects will now promote faster, resulting in your Old Gen growing faster. Given more NewGen, we are able to slow down the intervals of Young Gen collections, resulting in more time for short lived objects to die young and overall more efficient GC behavior.
4. **G1MixedGCLiveThresholdPercent**: Controls when to include regions in Mixed GC's in the Young GC collection, keeping Old Gen tidy without doing a normal Old Gen GC collection. When your memory is less than this percent, old gen won't even be included in 'mixed' collections. Mixed are not as heavy as a full old collection, so having small incremental cleanups of old keeps memory usage light.
   Default is 65 to 85 depending on Java Version, we are setting to 90 to ensure we reclaim garbage in old gen as fast as possible to retain as much free regions as we can.My Old flag set had this at 35 which was a bug. I had the intent of this flag inverted, as I thought 35 was what 65 does. You should not be using 35 for this number.
5. **G1ReservePercent=20:** MC Memory allocation rate in up to date versions is really insane. We run the risk of a dreaded "to-space exhaustion" not having enough memory free to move data around. This ensures more memory is waiting to be used for this operation. Default is 10, so we are giving another 10 to it.

6. **MaxTenuringThreshold=1**: Minecraft has a really high allocation rate of memory. Of that memory, most is reclaimed in the eden generation. However transient data will overflow into survivor. Initially played with completely removing Survivor and had decent results, but does result in transient data making its way to Old which is not good.Max Tenuring 1 ensures that we do not promote transient data to old generation, but anything that survives 2 passes of Garbage Collection is just going to be assumed as longer-lived. Doing this greatly reduces pause times in Young Collections as copying data up to 15 times in Survivor space for a tenured object really takes a lot of time for actually old memory. Ideally the GC engine would track average age for objects instead and tenure out data faster, but that is not how it works.
   Considering average GC rate is 10s to the upwards of minutes per young collection, this does not result in any 'garbage' being promoted, and just delays longer lived memory to be collected in Mixed GC's.
7. **SurvivorRatio=32:** Because we drastically reduced MaxTenuringThreshold, we will be reducing use of survivor space drastically. This frees up more regions to be used by Eden instead.
8. **AlwaysPreTouch**: AlwaysPreTouch gets the memory setup and reserved at process start ensuring it is contiguous, improving the efficiency of it more. This improves the operating systems memory access speed. Mandatory to use Transparent Huge Pages
9. **+DisableExplicitGC:** Many plugins think they know how to control memory, and try to invoke garbage collection. Plugins that do this trigger a full garbage collection, triggering a massive lag spike. This flag disables plugins from trying to do this, protecting you from their bad code.
10. **MaxGCPauseMillis**=200: This setting controls how much memory is used in between the Minimum and Maximum ranges specified for your New Generation. This is a "goal" for how long you want your server to pause for collections. 200 is aiming for at most loss of 4 ticks. This will result in a short TPS drop, however the server can make up for this drop instantly, meaning it will have no meaningful impact to your TPS. 200ms is lower than players can recognize.In testing, having this value constrained to an even lower number results in G1 not recollecting memory fast enough and potentially running out of old gen triggering a Full collection. Just because this number is 200 does not mean every collection will be 200. It means it can use up to 200 if it really needs it, and we need to let it do its job when there is memory to collect.
11. **+ParallelRefProcEnabled**: Optimizes the GC process to use multiple threads for weak reference checking. Not sure why this isn't default….
12. G1RSetUpdatingPauseTimePercent=5: Default is 10% of time spent during pause updating Rsets, reduce this to 5% to make more of it concurrent to reduce pause durations.
13. G1MixedGCCountTarget=4: Default is 8. Because we are aiming to collect slower, with less old gen usage, try to reclaim old gen memory faster to avoid running out of old.
14. G1HeapRegionSize=8M+: Default is auto calculated. SUPER important for Minecraft, especially 1.15, as with low memory situations, the default calculation will in most times be too low. Any memory allocation half of this size (4MB) will be treated as "Humongous" and promote straight to old generation and is harder to free. If you allow java to use the default, you will be destroyed with a significant chunk of your memory getting treated as

Humongous.

15. +PerfDisableSharedMem: Causes GC to write to file system which can cause major latency if disk IO is high – See https://www.evanjones.ca/jvm-mmap-pause.html

Using Large Pages

Also for Large Pages – It's even more important to use -Xms = -Xmx! Large Pages needs to have all of the memory specified for it or you could end up without the gains. This memory will not be used by the OS anyways, so use it.
Additionally use these flags (Metaspace is Java 8 Only, don't use it for Java7):

Code:

```
-XX:+UseLargePagesInMetaspace
```

Transparent Huge Pages

Controversial Feature but may be usable if you can not configure your host for real HugeTLBFS. try adding -XX:+UseTransparentHugePages but it's extremely important you also have AlwaysPreTouch set. Otherwise THP will likely hurt you. I have not measured how THP works for MC or its impact with AlwaysPreTouch, so this section is for the advanced users who want to experiement.

Credits:

Thanks to https://product.hubspot.com/blog/g1gc-fundamentals-lessons-from-taming-garbage-collection for helping reinforce my understanding of the flags and introduce improvements!

Changelog

- 5/2/2020: Added +PerfDisableSharedMem, Adjusted MixedGCTarget to 4
- 4/25/2020: Removed OmitStackTraces since it could cause performance issues with some plugins (but not everyone)
- **4/5/2020**: Massive refactor of the flag suggestions. Takes a new approach at optimizing pause times. Flags may still be changing. These changes are mandatory for MC 1.15
- **10/4/2018**: Removed AggressiveOpts and InitiatingHeapOccupancyPercent. Aggressive is removed in Java 11, and IHOP may hurt performance in Java 11. You should remove them for Java 8 too.

- **8/18/2018**: Adjusted MixedGCLiveThreshold to 35 (from 50) to ensure mixed GC’s start earlier.
  Added notes about recommended use of 10GB of memory.
  Added more flag documentation
- **5/24/2018**: Added -XX:+ParallelRefProcEnabled

Puffer.host

How to Optimize Your Minecraft Server with Aikar's Flags

Aikar's flags are a great way to optimize your Minecraft server's performance, and will allow you to squeeze out every last drop of performance from your CPU and RAM. In this article, we will discuss what Aikar's flags are, how they work, and how you can apply them to your server.

How to apply Aikar's Flags

Applying Aikar's flags is as simple as updating your Minecraft server's startup script to use the following Java flags:

```
java -Xms4096M -Xmx4096M -Dterminal.jline=false -Dterminal.ansi=true -XX:+UseG1GC -XX:+ParallelRefProcEnabled -XX:MaxGCPauseMillis=200 -XX:+UnlockExperimentalVMOptions -XX:+DisableExplicitGC -XX:+AlwaysPreTouch -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=4 -XX:G1MixedGCLiveThresholdPercent=90 -XX:G1RSetUpdatingPauseTimePercent=5 -XX:SurvivorRatio=32 -XX:+PerfDisableSharedMem -XX:MaxTenuringThreshold=1 -XX:G1NewSizePercent=30 -XX:G1MaxNewSizePercent=40 -XX:G1HeapRegionSize=8M -XX:G1ReservePercent=20 -XX:InitiatingHeapOccupancyPercent=15 -Dusing.aikars.flags=https://mcflags.emc.gs -Daikars.new.flags=true -jar server.jar nogui
```

You will need to make a few simple modifications to the flags before you are ready to start your server. First, you will need to change the amount of RAM allocated to your server in the first two flags. In the example, we are allocating 4096MB of memory, but you may want to allocate more or less. We don't recommend allocating less than 2GB (2048MB) of memory to your server. We have an entire guide on choosing the right amount of memory for your Minecraft server that you can read if you want more information.

The second flag that you will need to modify is at the very end, where you will need to specify your server's jar file. In this example, we named the jar file `server.jar`, but you will need to update this to your server's jar file name, or rename your server's jar file to match `server.jar`.

Once you have applied these flags, you can start your server. Some hosting companies may not allow you to use Aikar's flags on your server. Pufferfish Host provides servers with Aikar's flags enabled by default, so your server is always as optimized as possible.

Special Considerations

If you are using Pterodactyl Panel (or otherwise running your server in a resource-limited environment), you will need to leave some room for overhead. If you limit your server to 16GB, then you will only want to allocate 12GB or 14GB in Aikar's flags. This is due to various overhead that the JVM incurs when running using Aikar's flags. The exact cause of this overhead is sometimes disputed but generally it can be attributed to native buffers, GC overhead, and in some rare cases, memory leaks. There is much fluctuation with the amount of overhead required, so if your server is crashing due to memory-related issues, you may need to allocate less memory in Aikar's flags. Pufferfish Host covers this overhead for you free-of-charge.

What are Aikar's Flags

Aikar's flags are an optimized set of flags that can increase your server's performance with little effort on your part. By default, Java's garbage collector is not optimized for running Minecraft servers, but Aikar's flags is a set of tuned flags that is specifically designed for running Minecraft servers. These flags will reduce your server's pause times due to garbage collection and will cause the JVM to avoid lengthy old-gen garbage collection cycles.

Additional Flags

If you are using the Pufferfish server software, you can add the additional flag `--add-modules=jdk.incubator.vector` to Aikar's flags, anywhere before the `-jar` flag. This option will enable additional optimizations supported by the Pufferfish server software which rely on Java features which are still in beta. This is a great way to squeeze even more performance out of your Minecraft server.

Misconceptions

There are some common misconceptions around the usage of startup flags like Aikar's flags for Minecraft servers. First, using new flags like ZGC is not advisable. Performance on Aikar's flags is just fine with proper optimization, and switching to anything else is not likely to yield any improved performance results, and may even lead to decreased stability. Secondly, allocating too much memory can be detrimental to operating a server. We don't recommend ever allocating more than 16GB to a single Minecraft server, as this can cause the JVM's garbage collector to become congested. This can lead to increased GC pause times and will slow down your server. Only allocate as much memory as you need, your goal should be to allocate as little memory as possible.

Finishing Remarks

Aikar's flags are a great addition to any Minecraft server struggling with performance issues. If you have applied Aikar's flags (or if your host won't let you apply Aikar's flags) and your server is still having performance issues, consider switching to Pufferfish Host to upgrade your hardware. The greatest performance gains often come from simply using more capable hardware. If you are unable to switch hosting providers, you can consider switching to our custom server software, which we make available to the community free-of-charge.

SystemD

The two files below are the merger of other Minecraft SystemD service files I came across on the web and the screen-based method I originally used for running Minecraft instances.

Here's a breakdown of the processes that facilitate automatically loading the Minecraft server instance(s) when the system boots up:

1. The systemd minecraft@ instance specific **service** launches **screen**. (ie: `systemctl start minecraft@creative1`)
2. **screen** launches the script **start-minecraft-server** (as instructed by the SystemD service).
3. **start-minecraft-server** launches **`java fabric-server-launch.jar`** (with a lot of other arguments for optimization)
4. **java** loads fabric-server-launch.jar
5. fabric-server-launch.jar takes care of loading your Minecraft server instance
6. You can access your screened server with `screen -rd minecraft-creative1` (or whatever name you used)

I chose to use screen a bash script instead of launching java directly from systemd for these two reasons:

1. **screen** allows you to directly interact with the Minecraft server console without needing an rcon application, and its the way I'm used to dealing with server instances.
2. Using a bash script to launch the Minecraft server instance was easier to deal with per-instance customizations. You can facilitate the same thing directly from systemd, I just didn't want to take the time to learn how to do.

Setting up your environment

Below is an example of the files and folders that are needed for this setup.

Prep commands

```
# create base folder
mkdir -p /opt/minecraft/server-instances

# create specific instance folder
INSTANCENAME=creative1
mkdir -p /opt/minecraft/server-instances/$INSTANCENAME

# change ownership if desired
```

```
chown -R minecraftuser:minecraftuser $MCROOT/server-instances

# create the systemd minecraft service runner
touch /etc/systemd/system/minecraft@.service
# copy the script contents from below into the file and save
vi /etc/systemd/system/minecraft@.service

# reload the systemd daemon
systemctl daemon-reload

# example: creative1 instance
touch /opt/minecraft/server-instances/creative1/start-minecraft-server
# copy the script contents from below into the file and save
vi /opt/minecraft/server-instances/creative1/start-minecraft-server
```

/etc/systemd/system/minecraft@.service

```
[Unit]
Description=Minecraft Server in a Screen - %i
Wants=network-online.target
After=network-online.target
ConditionPathExists=/opt/minecraft/server-instances/%i

[Service]
Type=forking

User=minecraftuser
Group=minecraftuser

WorkingDirectory=/opt/minecraft/server-instances/%i

ExecStart=/usr/bin/screen -dmS minecraft-%i ./start-minecraft-server
ExecStop=/usr/bin/screen -S minecraft-%i -X stop

Restart=on-failure
RestartSec=60s

[Install]
WantedBy=multi-user.target
```

/opt/minecraft/server-instances/%i/start-minecraft-server

The systemd service launches screen and tells it to run this script file per instance.

```
#!/bin/bash

# super basic command line example
#java -Xms4096M -Xmx4096M -jar fabric-server-launch.jar nogui
#java -Xms6144M -Xmx6144M -jar fabric-server-launch.jar nogui

# 20211210
# Added the following line to fix a security bug announced today
# -Dlog4j2.formatMsgNoLookups=true

# Command line based on aikars suggested:
/opt/jvm/temurin/jdk-21/bin/java -Xms8192M -Xmx8192M --add-modules=jdk.incubator.vector -
XX:+UseG1GC -XX:+ParallelRefProcEnabled -XX:MaxGCPauseMillis=200 -
XX:+UnlockExperimentalVMOptions -XX:+DisableExplicitGC -XX:+AlwaysPreTouch -
XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=4 -XX:InitiatingHeapOccupancyPercent=15 -
XX:G1MixedGCLiveThresholdPercent=90 -XX:G1RSetUpdatingPauseTimePercent=5 -XX:SurvivorRatio=32
-XX:+PerfDisableSharedMem -XX:MaxTenuringThreshold=1 -
Dusing.aikars.flags=https://mcflags.emc.gs -Daikars.new.flags=true -XX:G1NewSizePercent=30 -
XX:G1MaxNewSizePercent=40 -XX:G1HeapRegionSize=8M -XX:G1ReservePercent=20 -jar fabric-server-
launch.jar --nogui
```

SystemD systemctl commands

```
# systemctl commands for server-instance creative1
systemctl enable minecraft@creative1.service
systemctl start minecraft@creative1.service
systemctl status minecraft@creative1.service
systemctl stop minecraft@creative1.service
```

screen commands

```
# list all screens
screen -list
```

```
# join screen for instance creative1
screen -rd minecraft-creative1
```

#end

Atlauncher

Running with Wayland

```
# from /usr/share/applications/atlauncher.desktop
Exec=env _JAVA_AWT_WM_NONREPARENTING=1 atlauncher
```

Linux Mint Cinnamon - HighDPI fix

```
# from /usr/share/applications/atlauncher.desktop
Exec=env GDK_DPI_SCALE=0.5 GDK_SCALE=2 atlauncher
```

mcaselector on Linux

Download the proper version of JavaFX and then run with a command line similar to the one below:

```
java --module-path openjfx-21.0.7_linux-x64_bin-sdk/javafx-sdk-21.0.7/lib --add-modules ALL-MODULE-PATH -jar mcaselector-2.5.2.jar
```

I was successful running mcaselector on Ubuntu Sway Remix, but could not access the menus.

I had the same experience using the instructions here using the instructions for Linux with the JRE-FX-21 option. I also tried using JRE-FX-24 with the same results.

-end