

# Fine-Tuning Clearspace Performance

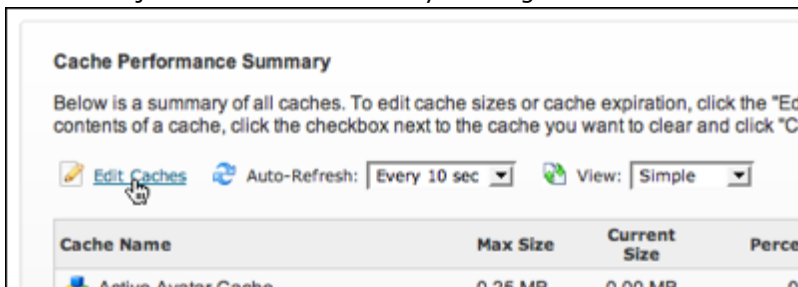
It's almost certain that you'll want to adjust Clearspace system settings from their defaults shortly after you're up and running. In particular, you'll want to keep an eye on caching, but there are other things you can do to ensure that the application is performing as well as possible.

## Adjusting Caches

Adjusting cache settings is the best way to improve performance. Clearspace uses a set of caches to keep performance high by caching data so that pages can be rendered quickly without directly querying the database.

## How Caches Work

- Each of nearly 100 kinds of data is cached in a separately adjustable cache. Each of these caches contains data that Clearspace might need to retrieve and display for users.
- You set each cache's maximum size (in memory) based on how much data might need to be cached there. If the maximum is reached during use, then Clearspace can no longer store data in the cache and must make queries to the database to retrieve the data the user has requested (by, say, viewing a particular page).
- A cache's performance is measured in terms of **effectiveness**. Effectiveness is a percentage that represents how often Clearspace attempts to get useful data from the cache and is able to. When the cache's maximum size is too small, Clearspace must query the database instead.
- Default maximum sizes for the caches are usually not appropriate for a new instance because circumstances vary so greatly from instance to instance. You almost always need to adjust.
- You can adjust cache maximums by clicking the **Edit Caches** link at the top of the **Caches** page.



## How to Fine Tune Caches

- Start by choosing a caches preset that makes sense for your use. When you're editing caches, you can choose one of the presets included with Clearspace: small, medium, and large (there's also a custom option). Each of these sets all caches to particular default levels. A good rule of thumb is to start with a larger preset, such as medium or large, then adjust downward if it turns out you don't need that much.
- Watch for caches with poor effectiveness and adjust those caches by increasing their maximum size. Clearspace signals especially poor effectiveness with a red box, as shown in the following illustration. The columns, left to right, are maximum size, current size, percent used, and effectiveness.

0.23 MB	0.00 MB	0.2%	41.0%
2.00 MB	0.23 MB	11.4%	97.4%
4.00 MB	3.83 MB	95.7%	59.1%
Unlimited	0.00 MB	0.0%	100.0%
2.00 MB	0.07 MB	2.7%	99.5%

- Use the Auto-Refresh interval on the cache performance summary page to have the page refresh regularly with current numbers.
- For the first few weeks in production keep a close watch for low cache effectiveness (note the red boxes!). When effectiveness gets too low for a particular cache, increase its maximum size. Here's a suggested schedule:
  - Watch effectiveness hourly immediately after launch.
  - After caches become stable, watch daily.
  - On an ongoing basis, watch weekly.
- In general, try to keep effectiveness over 90 percent.
- Look for caches you can set to very low levels, such as those for Clearspace features you're not using. For example, if you're not using blogs, you might set blog-related caches to 0.5MB. (Don't set it to 0, though, which effectively sets it to "unlimited.") This frees up overall memory you can use for other cache maximums.
- Ensure that the total of cache maximums doesn't exceed one-third of the maximum memory you've assigned to the JVM. The total is shown at the bottom of the cache summary list, while your total Java memory is shown at the very bottom of the page where the Java memory monitor is displayed. See below for more information on adjusting the JVM settings.

## Adjusting Java Virtual Machine (JVM) Settings

As with any Java-based web application, you can sometimes improve performance by assigning particular values to Java Virtual Machine options. You do this typically by editing the start script that you use to start the application server. For example, for Apache Tomcat you can set the JAVA\_OPTS value in the catalina.sh or catalina.bat file; for the Clearspace standalone distribution, you edit this value in the start-clearspace.sh or start-clearspace.bat file. While options can vary among VM distributions (not all servers use the HotSpot VM), most of the same options are available across VMs.

Here's a line from a Tomcat start script that shows example values for JVM options:

```
JAVA_OPTS="-server -Djava.awt.headless=true -Xmx1024m -XX:+UseConcMarkSweepGC
-XX:+UseParNewGC -Dsun.rmi.dgc.client.gcInterval=3600000 -XX:MaxPermSize=128m
-XX:+PrintGCDetails -Xloggc:/path/to/log/file"
```

Here are some of the JVM adjustments you might make:

- Use the **-server** option. This enables the server mode for garbage collection, which reduces the number of garbage collections and increases overall application throughput. There is also a difference between minor and full garbage collections in this mode. Minor garbage collections clean up more volatile objects and take a short amount of time. Full garbage collections take longer but run over the entire heap to clean up more memory.
- Set **-Djava.awt.headless=true** to ensure that code requiring AWT will run on headless servers.
- Turn on **GC logging** using the following options: `XX:+PrintGCDetails` and `-Xloggc:/path/to/log/file`.
- **Optimize memory**. For multi-CPU servers: `-XX:+UseConcMarkSweepGC -XX:+UseParNewGC`
- Set the maximum memory, also known as **maximum heap size**, by using the `-Xmx` option. For

small sites (50 users or fewer), start with `-Xmx512m`; for medium sites use `-Xmx1024m`; for enterprise sites use `-Xmx2048m`. Don't set this to more than 2GB memory on a 32-bit machine (although you might experiment with more on a 64-bit machine). You should specify the heap size for the JVM based on the application's expected memory usage. When maximum heap size is too small, you might see `OutOfMemoryError` or an excessive number of garbage collections.

- Don't set minimum memory (the `-Xms` option). The minimum heap size is less important because it is only useful at startup.
- Increase the **PermGen maximum size** with a setting such as `-XX:MaxPermSize=128m`. This is a good next step if increasing the maximum heap size doesn't seem to be helping performance. The JVM stores class metadata such as Method and Class objects in a part of the heap called PermGen. When you're using the `-server` option (see above), the default is typically 64 MB. If you see unexplained `OutOfMemoryError` after adjusting the maximum heap size, try increasing the `MaxPermSize` to 128 MB. Note that the increased capacity will only be used if it is needed by the JVM.

For more on JVM options, see [Java HotSpot VM Options](#) or the documentation for the VM you're using.

## Other Ways to Tune

- Consider turning off threaded discussions.
- If you're using the Apache web server, have it record web access log data. Turn off access logging in the servlet container, which is duplicate data.
- On starting up the servlet container, execute the following command to eliminate any hung search lock files: `rm -rf /tmp/lucene*`
- If you're using a load balancer, make sure it's configured for session affinity/sticky sessions.
- Make sure your servlet container isn't checking for updated files too frequently. For example, in the Resin server, you can set `dependency-check-interval` to 1800s.