

Fine-Tuning Performance

Through adjustments to caches, JVM settings, and more, you can make sure that Clearspace is performing well.

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. Here's a list of what's included:

[Adjusting Caches](#) (page 1)

[Client-Side Resource Caching](#) (page 2)

[Server-Side Page Caching](#) (page 3)

[Recommendations for High-Traffic Sites](#) (page 4)

[Adjusting Java Virtual Machine Settings](#) (page 5)

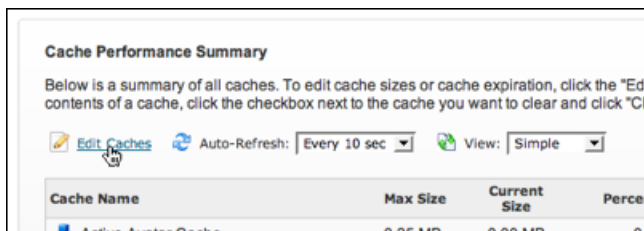
[Other Ways to Tune](#) (page 6)

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 Clearspace Uses Caches to Avoid Database Requests

- 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.

Maximum Size	Current Size	Percent Used	Effectiveness
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%
0.25 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.

Client-Side Resource Caching

You can guide client-side caching for static resources. These include images, scripts, stylesheets and so on. Caching these resources can help to improve application performance by reducing the number of calls to the Clearspace application server. The tradeoff is that cached data might be out of date. Note that this is for static content; for dynamic content (such as pages), headers are set by default to not cache.

You can set these with a few system properties in the admin console.

Property	Description	Values
<code>jive.maxAgeFilter.enabled</code>	Sets a header that prompts clients (such as browsers) to check the	true (default) to enable the filter; false to disable. Having this

Property	Description	Values
	age of their cached resource before requesting a new version.	enabled when you're testing isn't always useful because you usually want to bypass a cache for fresh content.
<code>jive.maxAgeFilter.maxage.seconds</code>	Sets the age after which a client-cached resource is considered stale and should be retrieved from the server.	A number of seconds. This defaults to 3600 (one hour). This sets <code>Cache-Control: max-age=3600</code> in the HTTP headers for static content.

Manually Enabling Client-Side Resource Caching

If you're using a version of Clearspace (prior to 2.0.1 and 1.10.6) in which headers supporting client-side caching aren't available, you can probably still configure your web server to set page headers to support client-side caching. For example, if you're using Apache 2.0, you can use the `mod_expires` module to have the server generate HTTP headers. To do this, add a configuration directive to your main server configuration. For more information on the `mod_expires` module, see the [Apache web site](#). Here's a configuration example:

```
# Enable expirations
ExpiresActive On
# CSS and JS files are good for two days from the time they were last accessed
ExpiresByType text/javascript "access plus two days"
ExpiresByType text/css "access plus two days"
```

Server-Side Page Caching

You can adjust server-side page caching for anonymous users when their having the very freshest content is less of a concern. With server-side caching on, the server caches pages that are assembled dynamically from data and resources. Retrieving a page from the cache can save the time needed to assemble a fresh page. However, if the data that makes up the page has changed, the page in the cache won't be as fresh as a new one would be.

With server-side page caching disabled (and for registered users, whether or not caching is enabled), Clearspace sends its default HTTP headers. With page caching enabled, in addition to the server-side page cache stored in memory, Clearspace also sets the HTTP header in the response to `Cache-Control max-age=3600`. The value set for `max-age` is configurable as described below.

You can set these with system properties in the admin console.

Property	Description	Values
<code>jive.pageCache.enabled</code>	Enables server-side page caching.	false (default) to disable page caching; true to enable it. When enabled, only anonymous or guest users will receive cached content.

Property	Description	Values
<code>jive.pageCache.maxage.seconds</code>	Sets the age after which the server will create a fresh page rather than retrieve the page from the cache.	Defaults to 60 seconds. This sets <code>Cache-Control: max-age=60</code> in the HTTP headers for the page.
<code>jive.pageCache.expiration.seconds</code>	Sets the number of seconds after which a page will be removed from the cache.	Defaults to 30 seconds
<code>jive.pageCache.maxEntries</code>	Sets the maximum number of pages that can be maintained in the cache. Note that increasing this value might require that you provide more system resources for Clearspace.	Defaults to 1000 entries.

Note that turning on developer mode by setting the `jive.devMode` property to true will disable the `maxAgeFilter` setting (effectively setting `jive.maxAgeFilter.enable` to false). The `jive.devMode` property is intended for situations when you're developing themes or plugins, In those situations, caching can hinder you from seeing the results of your development work.

Admin Console: System > Management > System Properties

Recommendations for High Traffic Sites

Here are a few recommendations for high-traffic deployments in which most users are anonymous. Anonymous users tend to visit merely to read content rather than contribute. Because they're anonymous, by default there's very little about their experience that will be customized. As a result, you can typically afford to serve them cached content rather than freshly rendered dynamic content.

- Recognize anonymous users from the cookies Clearspace sets. These cookies are available in Clearspace 1.x versions as of 1.10.3 and in version 2.0.1, but are unavailable in version 2.0. If you're using a version in which they're not available, you should consider upgrading.
 - Clearspace sets a `jive.user.loggedIn` cookie whose value is true if the user is registered and logged in.
 - Use the `jive.server.info` cookie to preserve consistency between user requests. This cookie contains information about the server from which the user receive content. Its value might be:

```
serverName=myhost:serverPort=8080:contextPath=/clearspace:localName=myhost:localPort=8080:localAddr=127.0.0.1
```

- Offload page caching to dedicated cached-content servers. With large numbers of anonymous users, you should consider using a content delivery network (CDN) such as Akamai or Limelight to serve cached content to anonymous users.
- Avoid sending a max age page cache header back to the client if the anonymous user will be able to log in. If the browser caches the page, then the newly logged in user might not receive fresh dynamic content.
- Consider caching resources such as CSS, Javascript, and decorator image files, which tend to rarely change. In fact, consider caching these resources for days if you expect not to be deploying new versions of them.
- Use the Clearspace Query Stats feature to track database queries. The Query Stats page of the Clearspace admin console can display SQL statements executed, how often they're executed, and how long the query

takes. The number of queries you see when browsing your front pages should be low. For more information about this feature, see Examining Database Queries in the *System Administrators' Guide*.

- When you have a large percentage of anonymous users but the overall number of page views is smaller, use Clearspace's built-in server-side page caching. Starting with version 2.0.0, Clearspace can [cache rendered pages for anonymous users](#) (page 3) . You can enable the cache and set maximum age and expiration thresholds.

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

Here are a few more ways to get Clearspace running the most efficiently.

- Consider setting the default for threaded discussions to "flat." People will still be able to set thread mode their own views to "threaded," but setting the default will ensure the "flat" mode for new users.
- 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.
- If you don't have HTTP Keep-Alive enabled, considering enabling it on your web server. Keep-Alive causes the connection between client and server to remain open between requests. This can improve performance by reducing the time spent opening connections. On Apache 2.0, the KeepAlive directive is set to On by default. For more on Keep-Alive, see the [Apache web site](#).
- 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.
- You can have Clearspace compress HTML and CSS files to reduce bandwidth. This gzip compression is supported only on certain application servers (see your server's documentation). You can enable this in the admin console at System > Settings > Page Compression.
- On Oracle 11g, use the prepared statement cache to reduce database overhead.
- If you're using Tomcat, use the [Apache Portable Runtime \(APR\)](#) connector rather than the standard connector. The APR connector tends to be more stable and perform better.
- You can improve performance on Linux by setting the following kernel startup parameters in your kernel boot loader:

```
kernel /vmlinuz-2.6.22.12 ro root=LABEL=/ rhgb quiet acpi=off
```