Performance Tuning DSpace

The software DSpace relies on does not come out of the box optimized for large repositories. Here are some tips to make it all run faster.

Bare Minimum Requirements

As of this writing, DSpace 7 is likely to require 4GB of memory at a *bare minimum*. However, with that little memory, you may quickly hit memory issues with any significant user activity or bulk uploading. So, *we recommend running DSpace with at least 8-12GB* (or more for very large or very active sites).

This minimum would roughly include...

- 2GB of memory for the Frontend (UI) / Node.js. Highly active sites will need more.
- 1GB of memory for the Backend (REST API) / JVM / Tomcat. Highly active sites will need more.
- 512MB of memory for PostgreSQL database. Highly active sites will need more.
- 512MB of memory for Solr. Highly active sites may need more.
- Extra memory may be required for command line scripts (which get kicked off in a separate JVM)

Keep in mind, because the frontend & backend can be run on separate servers, you can split this memory across two (or more) servers. You can even choose to run PostgreSQL or Solr either alongside the backend or on their own dedicated server.

The DSpace frontend (UI) will often require several CPUs, especially if you wish to use "cluster mode" (see below) to better scale your application. A smaller application may be able to use 4-6 CPU cores, while highly active sites may require additional CPU power. CPU is most often necessary for the frontend's Angular Serve Side Rendering (again see "cluster mode" notes below) and for any batch processing / command line scripts on backend.

Performance Tuning the Frontend (UI)

Use "cluster mode" of PM2 to avoid Node.js using a single CPU

If you are using PM2 to run the User Interface, you may want to start it using PM2's "Cluster Mode". This allows Node.js applications to be scaled across multiple CPUs by using the Node.js cluster module. See the PM2 Cluster Mode documentation at https://pm2.keymetrics.io/docs/usage/cluster-mode/

There are two ways to enable cluster mode. Choose one.

1. First, is by adding the "exec_mode" and "instances" settings to your JSON configuration as follows. You also may want to set the "max_memory_restart" option to avoid PM2 using too much memory. These three settings are described in more detail below. NOTE: make sure to start (or restart) your site to enable these settings (e.g. `pm2 start dspace-ui.json`)

dspace-ui.json

```
{
    "apps": [
        {
           "name": "dspace-ui",
           "cwd": "/full/path/to/dspace-ui-deploy",
           "script": "dist/server/main.js",
           "instances": "max",
           "exec_mode": "cluster",
           "env": {
              "NODE_ENV": "production"
           },
           "max_memory_restart": "500M"
        }
    ]
}
```

   a. Setting "exec_mode" to "cluster" will enable cluster mode,
   b. The "instances" setting allows you to customize how many CPUs are available to PM2 ("max" = all CPUs. But you also can specify a number like "8" = 8 CPUs.)
   c. The "max_memory_restart" setting is *optional* but tells PM2 how much memory to allow **per instance.** The example above has a maximum of 500MB. If the number of 'instances' is 8, that would mean PM2 could use up to 8 x 500MB = 4GB of memory. Therefore, you may wish to modify the values of "instances" and/or "max_memory_restart" to better control the memory available to PM2.

2. Alternatively, you can use command line flags to specify the same settings described above. The "-i" flag enables cluster mode and specifies the number of instances. The "--max-memory-restart" flag limits the memory per instance.

```
# Start the "dspace-ui" app. Cluster it across all available CPUs with a maximum memory of 500MB per CPU.
# This command is equivalent to the example cluster settings in the "dspace-ui.json" file above.
pm2 start dspace-ui.json -i max --max-memory-restart 500M
```

Give Node.js more memory

On machines with >2GB of memory available, Node will only use a maximum of 2GB of memory by default (see https://github.com/nodejs/node/issues/28202). This 2GB of memory should be enough to build & run the User Interface, but it's possible that highly active sites may require 4GB or more.

If you want to increase the memory available to Node.js, you can set the NODE_OPTIONS environment variable:

```
# Increase memory limit to 4GB (4096MB) by setting "max-old-space-size"
# in your NODE_OPTIONS environment variable
export NODE_OPTIONS=--max-old-space-size=4096
```

Turn on (or increase) caching of Server-Side Rendered pages

As of DSpace 7.5, we now provide basic, in-memory caching of server-side rendered (SSR) pages. Server-side rendering is used to pre-generate full HTML pages to pass back to users (primarily anonymous users and bots). This is necessary for Search Engine Optimization (SEO) as some web crawlers cannot use Javascript. It also can be used to immediately show the first HTML page to users while the Javascript app loads in the user's browser.

While server-side-rendering is highly recommended on all sites, it can result in Node.js having to pre-generate many HTML pages at once when a site has a large number of simultaneous users/bots. This may cause Node.js to spend a lot of time processing server-side-rendered content, slowing down the entire site.

Therefore, DSpace provides some basic caching of server-side rendered pages, which allows the same pre-generated HTML to be sent to many users/bots at once & decreases the frequency of server-side rendering.

These settings are documented at User Interface Configuration: Cache Settings - Server Side Rendering (SSR)

Performance Tuning the Backend (REST API)

Give Tomcat More Memory

Give Tomcat More Java Heap Memory

Java Heap Memory Recommendations

At the time of writing, DSpace recommends you should give Tomcat >= 512MB of Java Heap Memory to ensure optimal DSpace operation. Most larger sized or highly active DSpace installations however tend to allocate more like 1024MB (1GB) to 2048MB (2G) or more of Java Heap Memory.

Performance tuning in Java basically boils down to memory. If you are seeing "`java.lang.OutOfMemoryError: Java heap space`" errors, this is a sure sign that Tomcat isn't being provided with enough Heap Memory.

Tomcat is especially memory hungry, and will benefit from being given lots of RAM. To set the amount of memory available to Tomcat, use either the `JAVA_OPTS` or `CATALINA_OPTS` environment variable, e.g:

```
CATALINA_OPTS=-Xmx512m -Xms512m
```

OR

```
JAVA_OPTS=-Xmx512m -Xms512m
```

The above example sets the maximum Java Heap memory to 512MB.

Difference between JAVA_OPTS and CATALINA_OPTS

You can use either environment variable. `JAVA_OPTS` is also used by other Java programs (besides just Tomcat). `CATALINA_OPTS` is *only used* by Tomcat. So, if you only want to tweak the memory available to Tomcat, it is recommended that you use `CATALINA_OPTS`. If you set **both** `CATALINA_OPTS` and `JAVA_OPTS`, Tomcat will default to using the settings in `CATALINA_OPTS`.

If the machine is dedicated to DSpace a decent rule of thumb is to give tomcat half of the memory on your machine. **At a minimum, you should give Tomcat >= 512MB of memory for optimal DSpace operation.** (*NOTE: As your DSpace instance gets larger in size, you may need to increase this number to the several GB range.*) The latest guidance is to also set `-Xms` to the same value as `-Xmx` for server applications such as Tomcat.

Give Tomcat More Java PermGen Memory

Java PermGen Memory Recommendations

At the time of writing, DSpace recommends you should give Tomcat >= 128MB of PermGen Space to ensure optimal DSpace operation.

If you are seeing "`java.lang.OutOfMemoryError: PermGen space`" errors, this is a sure sign that Tomcat is running out PermGen Memory. (More info on PermGen Space: https://frankkieviet.blogspot.com/2006/10/classloader-leaks-dreaded-permgen-space.html)

To increase the amount of PermGen memory available to Tomcat (default=64MB), use either the `JAVA_OPTS` or `CATALINA_OPTS` environment variable, e.g:

```
CATALINA_OPTS=-XX:MaxPermSize=128m
```

OR

```
JAVA_OPTS=-XX:MaxPermSize=128m
```

The above example sets the maximum PermGen memory to 128MB.

Difference between JAVA_OPTS and CATALINA_OPTS

You can use either environment variable. `JAVA_OPTS` is also used by other Java programs (besides just Tomcat). `CATALINA_OPTS` is *only used* by Tomcat. So, if you only want to tweak the memory available to Tomcat, it is recommended that you use `CATALINA_OPTS`. If you set **both** `CATALINA_OPTS` and `JAVA_OPTS`, Tomcat will default to using the settings in `CATALINA_OPTS`.
Please note that you can obviously set **both** Tomcat's Heap space and PermGen Space together similar to:
`CATALINA_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m`
On an Ubuntu machine (10.04) at least, the file `/etc/default/tomcat6` appears to be the best place to put these environmental variables.

Choosing the size of memory spaces allocated to DSpace Backend

psi-probe is a webapp that can be deployed in DSpace and be used to watch memory usage of the other webapps deployed in the same instance of Tomcat (in our case, the DSpace server webapp).

1. Download the latest version of psi-probe from https://github.com/psi-probe/psi-probe
2. Unzip probe.war into [dspace]/webapps/

```
cd [dspace]/webapps/
unzip ~/probe-3.1.0.zip
unzip probe.war -d probe
```

3. Add a Context element in Tomcat's configuration, and make it privileged (so that it can monitor the other webapps):
   EITHER in `$CATALINA_HOME/conf/server.xml`

   ```
   <Context docBase="[dspace]/webapps/probe" privileged="true" path="/probe" />
   ```

   OR in `$CATALINA_HOME/conf/Catalina/localhost/probe.xml`

   ```
   <Context docBase="[dspace]/webapps/probe" privileged="true" />
   ```

4. Edit `$CATALINA_HOME/conf/tomcat-users.xml` to add a user for logging into psi-probe (see more in https://github.com/psi-probe/psi-probe/wiki/InstallationApacheTomcat)

   ```
   <?xml version='1.0' encoding='utf-8'?>
   <tomcat-users>
     <user username="admin" password="t0psecret" roles="manager" />
   </tomcat-users>
   ```

5. Restart Tomcat
6. Open http://yourdspace.com:8080/probe/ (edit domain and port number as necessary) in your browser and use the username and password from tomcat-users.xml to log in.

In the "System Information" tab, go to the "Memory utilization" menu. Note how much memory Tomcat is using upon startup and use a slightly higher value than that for the `-Xms` parameter (initial Java heap size). Watch how big the various memory spaces get over time (hours or days), as you run various common DSpace tasks that put load on memory, including indexing, reindexing, importing items into the oai index etc. These maximum values will determine the `-Xmx` parameter (maximum Java heap size). Watching PS Perm Gen grow over time will let you choose the value for the `-XX:MaxPermSize` parameter.

Give the Command Line Tools More Memory

Give the Command Line Tools More Java Heap Memory

Similar to Tomcat, you may also need to give the DSpace Java-based command-line tools more Java Heap memory. If you are seeing "`java.lang.OutOfMemoryError: Java heap space`" errors, when running a command-line tool, this is a sure sign that it isn't being provided with enough Heap Memory.

By default, DSpace only provides 256MB of maximum heap memory to its command-line tools.

If you'd like to provide **more** memory to command-line tools, you can do so via the `JAVA_OPTS` environment variable (which is used by the `[dspace]/bin/dspace` script). Again, it's the same syntax as above:

```
JAVA_OPTS=-Xmx512m -Xms512m
```

This is especially useful for big batch jobs, which may require additional memory.

You can also edit the `[dspace]/bin/dspace` script and add the environmental variables to the script directly.

Give the Command Line Tools More Java PermGen Space Memory

Similar to Tomcat, you may also need to give the DSpace Java-based command-line tools more PermGen Space. If you are seeing "`java.lang.OutOfMemoryError: PermGen space`" errors, when running a command-line tool, this is a sure sign that it isn't being provided with enough PermGen Space.

By default, Java only provides 64MB of maximum PermGen space.

If you'd like to provide **more** PermGen Space to command-line tools, you can do so via the `JAVA_OPTS` environment variable (which is used by the `[dspace]/bin/dspace` script). Again, it's the same syntax as above:

```
JAVA_OPTS=-XX:MaxPermSize=128m
```

This is especially useful for big batch jobs, which may require additional memory.

Please note that you can obviously set **both** Java's Heap space and PermGen Space together similar to:
`JAVA_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m`

Give PostgreSQL Database More Memory

On many Linux distros PostgreSQL comes out of the box with an incredibly conservative configuration - it uses only 8Mb of memory! To put some more fire in its belly edit the `shared_buffers` parameter in `postgresql.conf`. The memory usage is 8KB multiplied by this value. The advice in the Postgres docs is not to increase it above 1/3 of the memory on your machine.

For More PostgreSQL Tips

For more hints/tips with PostgreSQL configurations and performance tuning, see also:

- PostgresPerformanceTuning
- PostgresqlConfiguration

Performance Tuning Solr

Solr has it's own detailed documentation with recommendations for "Taking Solr to Production". We recommend following the recommendations from Solr, especially related to "Ulimit settings" (for Unix-based systems) and "Avoiding Swapping" (for Unix-based systems). See the Solr documentation for more details.