Apache Cocoon Performance Tips (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Disclaimer

The Cocoon Performance Tips in this version is a loose collection of usenet articles regarding how to improve the Apache Cocoon performance.

As in the real world, it needs some kind of evolution to get better. If you have suggestions how to make it better or new kool tips, then be brave and send it to the Cocoon Mailing Lists!

Sometimes the tips maybe doubled or contradictory. If you notice something like that, then send a note to the Cocoon Mailing Lists.

2. Common

- Logging kills performance. Consider disabling logging entirely from Cocoon (leave only the ERROR channel) and let Apache or the servlet container log accesses and stuff.
- Use a transparent proxy in front of your web server! The fastest response is the one that is not even processed. Cocoon is very slow (compared to a proxy server) to read resources such as stylesheets and images. A transparent proxy (SQUID, for example, don't use Apache 1.3 mod_proxy because it is not fully compatible with HTTP/1.1 and disables connection keep-alive - Apache 2 mod_proxy is fine). Make sure you tune how long the static resources that Cocoon "read"s from the sitemap are cached (look into the readers code to find out more).
- Consider prerendering or time-based batch-process the static parts of your site. PDF reports, rasterized SVG graphs or things that change regularly.
- For optimum performance with Tomcat 4 and Cocoon 2, use the HTTP/1.0 connector.
- Move static content out of Cocoon's control. Move your static content out of the Cocoon servlet context and into its own context (just letting Tomcat serve directly). An even better approach would be to use a front-end webserver to serve the static, but installing Apache + Tomcat + our Cocoon app would be a bit much when Tomcat + our Cocoon app is doing fine.
- Disable resource reloading. The disk I/O system could become the bottleneck.
- Search for messages such as "decommissioning instance of...". This reveals some undersized pools which are corrected by tuning cocoon.xconf and sitemap.xmap. Undersized pools act like an object factory, plus the ServiceManager overhead.

3. Caching and Pooling

- Fine-tune the pool sizes for components in the files cocoon.xconf and sitemap.xmap. If the pools are too small for the load this will have a great impact on your performance. The goal is to achieve such a configuration that for every request there is a free component in the pool. Suppose, you have up to 100 simultaneous requests and your pipelines have up to 2 xslt transformers, then you need to set the maximum pool size to 200 xslt transformers. They will be created when needed and retained to the pool for future use.
- Fine-tune the Cocoon settings for the store and the other stuff.
- Important is the size of the documents that will be cached, because caching appears to be very time consuming process.
- Use the Caching Pipelines in your sitemaps where relevant.
- If your cache is set too small to keep the entire XML in memory, then the cache will be of no benefit.
- Watch the cachability in the log files, and make sure that things are being fed from the cache.

- Only use dynamic data when it is needed. Dynamic pages can't be cached 100%.
- Don't put Cocoon webapp too deep into directory structure. Cache keys contain absolute file names (or hash values of the absolute file names - in 2.0.X series), and the deeper cocoon is located in the filesystem, the longer keys are becoming. Obviously, longer keys will take more time to process them. In worst case scenario, slowdown up to 10% could be achieved (unscientific observations, do your own test).

More information about caching can be found in the Caching documentation. Especially look at the information about the expires configuration.

4. JVM and OS

- Consider using a good JVM on a good OS. Scalability is a very different beast than pure speed. An Apple DualG4 866 seems to run faster than a Sun Enterprise 4500 (and costs a fraction), but try hitting them with 2000 concurrent Cocoon requests.
- Fine-tune your JVM settings (max heap-size, initial memory, s.o.). Please read the Java Performance FAQ's and the Tuning Garbage Collection Document.
- Don't specify the -Xms parameter.
- Set the -Xnoclassgc parameter on the Sun JDK 1.3.1! It reduces the frequency of need for garbage collection by permitting the memory allocated to unused classes to be reused (instead of having to be collected and/or compacted). Less fragmentation means less collection means better response times.

5. Perfomance Formulas

- Consider following formula for Pipeline Processing:
  Number_of_simultaneous_users * depth_of_content_aggregation
- Consider following formula for Generators/Transformers/Serializers:
  Amount_required_to_process_one_request * Number_of_simultaneous_users
- Consider following formula for Connectors:
  Count_of_pipeline_components_to_process_one_request * Number_of_simultaneous_users

6. Pipelines

- Keep an eye on the overall complexity of pipelines.
- Try to keep the size of the documents going through the pipeline small. To big documents slows down translation.
- Use the Caching Pipelines in your sitemaps where relevant.
- Use the expires parameter (see above) as frequently as you can. It improves the end user experience dramatically. Browsers and intermediate proxy servers love the HTTP Expires header.

7. XSP

Consider turning your XSPs into Generators by hand and call them directly. Of course you don't need to do this for all pages, but it's recommended to it for those which are heavy loaded.

You can try it this way:

Cocoon will compile your XSP's into Java classes (see tomcat/work/..../org/apache/cocoon/www/my_xsp.class). After that, add the generated Generator to the Sitemap:
<map:generator type="myXSP" src="org.apache.cocoon.www.my_xsp"/>

And use it:
<map:generate type="myXSP"/>

8. XSLT and XSL

For more tips and information about XSL and XSLT grep the Internet and the Xalan Homepage

- Try to keep the number of templates in the XSL translation small.
- There are several ways of doing the same stuff in XSLT, test the difference between them.
- Consider browser-dependent targetting to perform client-side XSLT. Cocoon is very fast if it doesn't do transformations. IE 5.5 and 6 are pretty compliant and might be something around 30% of your hits (probably more on some popular public web sites like Nasa's). Reducing one/third of the transformations might speed up a LOT.
- How complicated are the XSLT stylesheets? If you are not using global variables or parameters this will speeds things up.
- Consider using XSLTC instead of Xalan. XSLTC compiles XSLT to bytecode (translets) the first time a stylesheet is used. Consequently it uses the compiled code which is faster by a magnitude than the interpreted one.

1. Comments

add your comments