CQ Performance Analysis: Optimizing Response Time

David Collie | Technical Account Manager

Topics Covered

- How to analyze CQ request/response performance
  - Using Request Log Analyzer Tool
  - Using Sling Request Analyzer Tool

- How to generate/analyze thread dumps and CPU profiling output
  - Jstack**,** Jconsole
  - TDA (Thread Dump Analyzer)

- Combining CQ & JVM methods to focus in on cause
  - A worked example

Perspectives from Users, Authors and Administrators

Users

“Pages are taking a long time to load.”
“I get 503 errors when accessing the site.”
“When I access a page nothing happens or it times out.”

Authors

“When I do action X, the page is slow”
“I’m unable to create a web page”
“The system breaks when I try to process X”

Administrators

“The Load Balancer logs are full of time out errors”
“Our system alerts are going off regularly under normal load”
“High response times on publisher tier”

Performance Bottlenecks

- Custom Application Code
  - + identify existing code, change implementation, optimize
- Third Party Calls (HTTP, LDAP, Database, Network Latency)
  - + investigate cause of latency/excessive reads, implement caching
- CQ Product
  - + change configuration, persistence, ask Product Team for help
- Java Virtual Machine
  - + change GC type, assign more/less heap memory, locks
- OS/ Hardware
  - + file system, add memory, faster disk reads, resource contention

Performance Bottlenecks

?

Where Do I Start

TL;DR

- If you only do one set of things, make them this!
- During performance problems:
  - Take 10 or more thread dumps (i.e. 5 secs apart)
  - Gather request.log and error.log
- Run following tools
  - rlog.jar the request.log
  - ‘tda’ the thread dumps
  - Visually inspect the error.log
- Analysis doesn’t show you the issue?
  - Ask the Daycare team for help!!! (Don’t forget Felix Configuration)

CQ Tools & Techniques

Tools & Techniques

- CQ Performance Tools
  - Logging
  - Request Log Analyzer
  - Sling Request Analyzer

- Java Performance Tools
  - Thread Dumps
  - Profiling

CQ Logging & Debug Settings

- Standard log files (crx-launchpad/logs)
  - request.log – request has an entry, when response is sent has entry
  - error.log – default capture file for any output from CQ/Application logging
  - access.log – traditional web server logging with UA string (not needed here)
- Custom logging configuration
  - Configured in the repository via sling:OsgiConfig node
  - Useful after determining a focus in investigating the issue (Rinse & Repeat)
- Page Rendering Logging
  - View page source in browser, gives a URL for Charting
  - Shows how long each part of the page took to render

CQ Request Log Analyzer

- Run the request log analyzer with this command:

```
$ java -jar crx-quickstart/opt/helpers/rlog.jar crx-quickstart/logs/request.log
```

- Understanding the output:
  - Sorted list of requests made to the CQ instance and their response times.
  - Sorted by slowest to fastest response time.
- Options of rlog.jar

```
Usage:
  java -jar rlog.jar [options] <filename>
Options:
  -h                Prints this usage.
  -n <maxResults>  Limits output to <maxResults> lines.
  -m <maxRequests> Limits input to <maxRequest> requests.
  -xdev             Exclude POST request to CRXDE.
```

CQ Request Log Analyzer

```
238157ms 14/Aug/2012:18:39:57 -0700 200 POST /bin/wcmcommand text/html

51652ms 15/Aug/2012:11:40:57 -0700 200 GET /libs/cq/core/content/welcome.html text/html; charset=utf-8

29515ms 04/Aug/2012:17:06:42 -0700 200 GET /content/geometrixx/en/services/banking.html text/html

104319ms 30/Jul/2012:14:30:25 -0700 200 GET /content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.mov -

81667ms 08/Aug/2012:19:48:36 -0700 200 GET /content/tests-performance/data/video/our_super_new_time_1080p_h264.backup.mov -

34304ms 08/Aug/2012:17:29:11 -0700 200 GET /content/tests-performance/data/video/another_type_of_video_1080p_h264.backup.mov -

20816ms 08/Aug/2012:17:29:11 -0700 200 GET /libs/cq/tagging/widgets.js application/x-javascript

20798ms 08/Aug/2012:17:29:11 -0700 200 GET /libs/cq/security/widgets.js application/x-javascript
```

Sling Request Processing Tracker

How to Configure the Sling Request Analyzer Tool

To setup the requesttracker.txt log:

1. Download org.apache.sling.reqanalyzer-0.0.1-SNAPSHOT.jar
2. Go to http://{host}:{port}/system/console/bundles
3. Upload and install the jar as an OSGi bundle

A log file will be generated under crx-quickstart/logs/requesttracker.txt

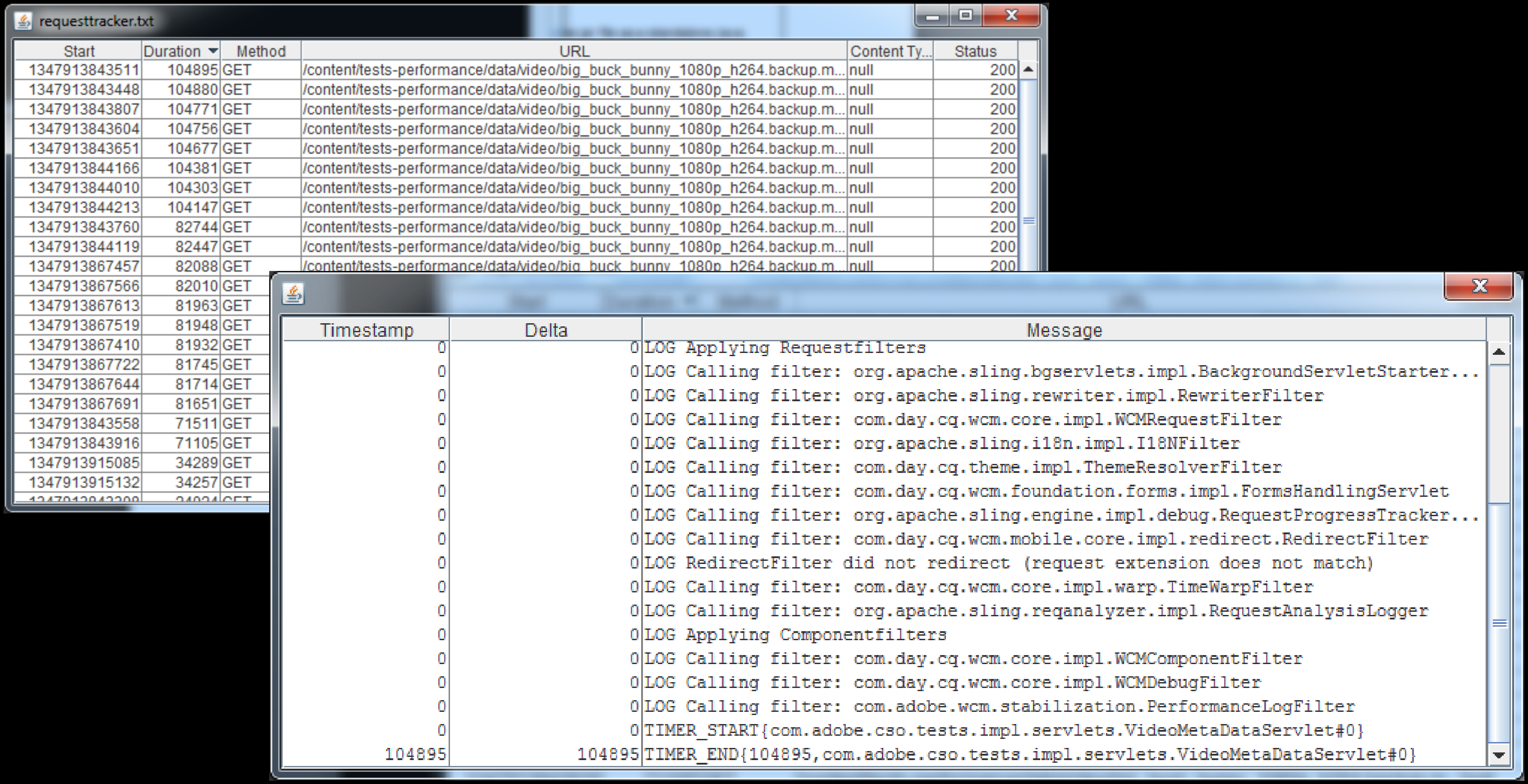
To analyze the requesttracker.txt file, use the same jar file as a standalone Java application.

Start the module using this java command:

```
$ java -jar org.apache.sling.reqanalyzer-0.0.1-SNAPSHOT.jar requesttracker.txt
```

Sling Request Processing Tracker

Open with a drillable view of requests:



requesttracker.txt

Start	Duration	Method	URL	Content Ty...	Status
1347913843511	104895	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913843448	104880	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913843807	104771	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913843604	104756	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913843651	104677	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913844166	104381	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913844010	104303	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913844213	104147	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913843760	82744	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913844119	82447	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913867457	82088	GET	/content/tests-performance/data/video/big_buck_bunny_1080p_h264.backup.m...	null	200
1347913867566	82010	GET			
1347913867613	81963	GET			
1347913867519	81948	GET			
1347913867410	81932	GET			
1347913867722	81745	GET			
1347913867644	81714	GET			
1347913867691	81651	GET			
1347913843558	71511	GET			
1347913843916	71105	GET			
1347913915085	34289	GET			
1347913915132	34257	GET			

Timestamp	Delta	Message
0	0	LOG Applying Requestfilters
0	0	LOG Calling filter: org.apache.sling.bgservlets.impl.BackgroundServletStarter...
0	0	LOG Calling filter: org.apache.sling.rewriter.impl.RewriterFilter
0	0	LOG Calling filter: com.day.cq.wcm.core.impl.WCMRequestFilter
0	0	LOG Calling filter: org.apache.sling.i18n.impl.I18NFilter
0	0	LOG Calling filter: com.day.cq.theme.impl.ThemeResolverFilter
0	0	LOG Calling filter: com.day.cq.wcm.foundation.forms.impl.FormsHandlingServlet
0	0	LOG Calling filter: org.apache.sling.engine.impl.debug.RequestProgressTracker...
0	0	LOG Calling filter: com.day.cq.wcm.mobile.core.impl.redirect.RedirectFilter
0	0	LOG RedirectFilter did not redirect (request extension does not match)
0	0	LOG Calling filter: com.day.cq.wcm.core.impl.warp.TimeWarpFilter
0	0	LOG Calling filter: org.apache.sling.reqanalyzer.impl.RequestAnalysisLogger
0	0	LOG Applying Componentfilters
0	0	LOG Calling filter: com.day.cq.wcm.core.impl.WCMComponentFilter
0	0	LOG Calling filter: com.day.cq.wcm.core.impl.WCMDebugFilter
0	0	LOG Calling filter: com.adobe.wcm.stabilization.PerformanceLogFilter
0	0	TIMER_START{com.adobe.cso.tests.impl.servlets.VideoMetaDataServlet#0}
104895	104895	TIMER_END{104895,com.adobe.cso.tests.impl.servlets.VideoMetaDataServlet#0}

JVM Tools & Techniques

Tools & Techniques

- CQ Performance Tools
  - Logging (Default Log Files, Adding Custom Logging)
  - Request Log Analyzer
  - Sling Request Processing Analyzer

- Java Performance Tools
  - Thread Dumps
  - Profiling

Thread Dumps

- In Linux or Unix:
  1. Look up the pid of the java process:

     ```
     $ ps -ef  | grep java
     ```

  2. Run either of these commands:
     - This will dump the thread dump to stdout of the CQ java process:

       ```
       $ kill -QUIT {pid}
       ```

     - This will dump the thread dump to the command prompt directly:

       ```
       $ sudo -u {cquser} $JAVA_HOME/bin/jstack {pid} >> threaddumps.txt
       ```

- For complete instructions (including those for Windows OS) go here:
  - http://dev.day.com/content/kb/home/cq5/CQ5SystemAdministration/TakeThreadDump.html

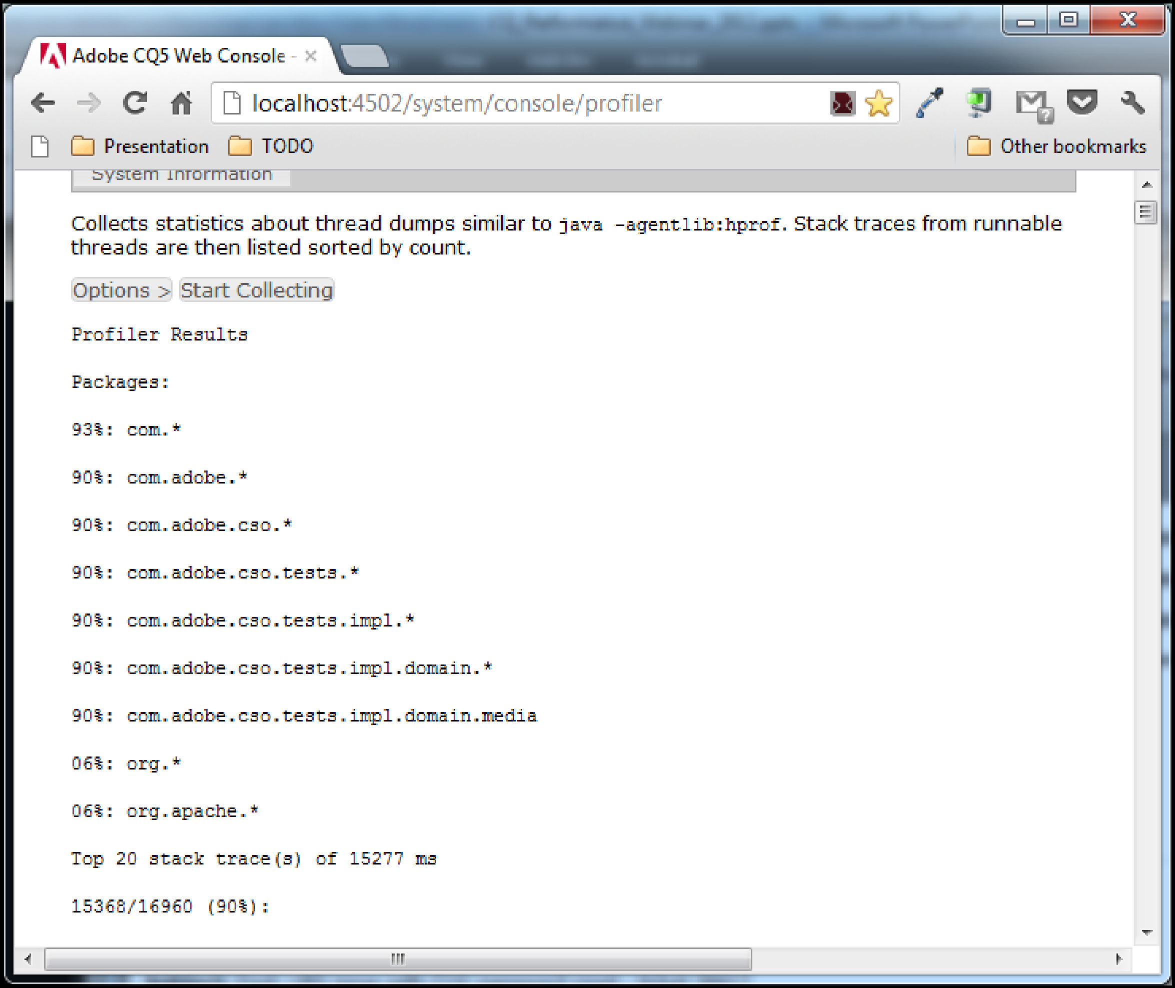
Thread Dumps

- Why take multiple thread dumps?
  - This gives us a view of the process over time.
  - See which threads are running for a long time, shows the interaction of threads
- How many thread dumps and at what interval?
  - 10 thread dumps at a 1-3 second interval
  - Sufficient for analyzing request threads, see some progression
- What are the bottlenecks that can be seen in a thread dump?
  - Greedy locks/monitors or excessive use of synchronized blocks.
  - Are many threads stuck doing any of the following?
    - In socketRead, waiting on a response from a remote web service?
    - In socketWrite. waiting on a remote web service to receive data we are sending?
    - Reading or writing to files? This may point to slow disk I/O or lack of application caching.

Profiling

- What does it do?
  - Takes a sample stack traces of RUNNABLE (running) threads at a given interval.
  - Runs a comparison to see which stacks show up most frequently.
- What is it good for?
  - Showing hotspots in code that are taking a lot of CPU time.
- What is it NOT good for?
  - Showing progression of threads over time.
  - Showing lock contention.
- See this article for details:
  - http://dev.day.com/content/kb/home/Crx/Troubleshooting/AnalyzeUsingBuiltInProfiler.html

Profiling



Collects statistics about thread dumps similar to `java -agentlib:hprof`. Stack traces from runnable threads are then listed sorted by count.

```
Profiler Results

Packages:

93%: com.*

90%: com.adobe.*

90%: com.adobe.cso.*

90%: com.adobe.cso.tests.*

90%: com.adobe.cso.tests.impl.*

90%: com.adobe.cso.tests.impl.domain.*

90%: com.adobe.cso.tests.impl.domain.media

06%: org.*

06%: org.apache.*

Top 20 stack trace(s) of 15277 ms

15368/16960 (90%):
```

Putting CQ & JVM Performance Tools Together

Worked Example

1. Use Jmeter to put load on an instance
   - Preconfigured with “badly” performing code

2. During load test:
   - run CQ Profiler during tests
   - take thread dumps with script

3. Gather log files from system:
   - Use RLOG tool – find slow resources
   - Use Sling Request Processor – find slow Java processes
   - Use TDA – identify blocked threads, long running threads

Get code and sample scripts from:
https://github.com/cqsupport/webinar-optimizingrequestperformance

Q& A + References

- CQ Tools & Documentation
  - Request Log Analyzer
    http://dev.day.com/docs/en/cq/current/howto/performance_monitor.html#Using%20rlog.jar%20to%20find%20requests%20with%20long%20duration%20times
  - CQ Profiler
    http://dev.day.com/content/kb/home/Crx/Troubleshooting/AnalyzeUsingBuiltInProfiler.html
  - Sling Request Processing Analyzer
    http://sling.staging.apache.org/documentation/bundles/request-analysis.html
  - Taking Thread Dumps
    http://dev.day.com/content/kb/home/cq5/CQ5SystemAdministration/TakeThreadDump.html
- Thread Dump Tools
  - TDA http://java.net/projects/tda/downloads
  - IBM Thread and Monitor Dump Analyzer:
    https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c
  - Samurai: http://yusuke.homeip.net/samurai/en/index.html

Adobe