ColdFusion Performance Analysis and Tuning
Brandon Purcell
Principal Support Engineer
Macromedia

Contributions by
Jim Schley &
Mike Nimer
ColdFusion MX Performance and Tuning

- Tuning Process Overview
- Tuning ColdFusion Applications
- Tuning ColdFusion Settings
- Tuning Application Server Settings
- Monitoring Performance
Tuning Process

- Measuring Performance
  - Response Time
  - Throughput

- Single-user baseline performance
Tuning Process

- Tuning cycle
  - Change one thing at a time
  - Load test & measure performance
  - Make note of the result
  - Repeat until optimum performance is reached

1. Change code or configuration
2. Load Test
3. Record Performance Metrics
Tuning Process

- Identify the tunables
  - Application Code
  - ColdFusion Settings
  - Application Server Settings
  - Web Server Settings
  - OS & Network Settings

- Identify Bottlenecks, fix, retest
Tuning Process: Load Testing

- Load testing is critical to web development and deployment process
- Essential tool for tuning
- Identifies
  - Very specific bottlenecks
  - Specific stability problems
- Answers:
  - “How many servers do we need?”
  - “Are we ready to go live?”
- Load test must be performed PRIOR to deployment
Testing During Development

- Write small tests scripts one for each feature of the application/site. (search, login, etc.)
- Run these tests with 5-10 stress users (no simulated wait) while you're still coding the functionality.
- These scripts should have no wait times, or should be run in stress mode.
Bottleneck Testing

- Use the simple scripts created during development testing.
- Look for slow running pages in the CF Debug output.
- Use a query analyzer to improve db queries
- Use the debug tree
- Use getTickCount() to find sections of slow running cfml
- Rule of Thumb: Pages that take more then 200-300ms with no load should be looked at for performance improvements
Full Site Testing

- Determine amount of traffic site hardware can support. Find your per-traffic load amounts so you know when to add new servers to the cluster.

- Write one script for each type of user
  - 1 test only hits the home page
  - 1 test will only browse the categories
  - 1 test will perform a search
  - 1 test will go through the checkout process

- Assign Percentages to each of these steps. If 60% of all users only hit your home page, then make sure your full site testing has 60% of it’s simulated users running the home page only test.
Full Site Testing (cont’d)

- Define a base line
- Establish environment “breaking point”
  - Server hangs/restarts
  - Page execution time exceeds threshold
- Perform Site Testing a Hardware that matches Production!
- Ensure accurate operation for multiple concurrent users
  - Testing session functionality in a clustered environment.
  - Complete test of database/backend system access
- Use these tests to do server tuning.
  - Adjust OS, WWW, CF Administrator settings to tune to application(s) for Response Times or Throughput
Tuning ColdFusion Applications

- Start with Application Code
- Common areas to look at
  - Database Queries
  - External Calls (Web Services, HTTP, Mail, CFX, etc)
  - Caching (reducing I/O)
  - Scoping
- Manage User Perception of Performance
Tuning ColdFusion Applications

- If “It’s Slow”
  - Identify slow templates/pages on site
  - Determine whether the DB queries are bottleneck
  - If queries are NOT the bottleneck, use `GetTickCount()` to track down slow running CFML or CFX bottleneck
  - Fix & retest
General Query Performance Tips

- Analyze your queries, using query analyzer tools
- Indexes – make sure you use them, but limit their usage on tables with heavy insert/update traffic.
- Limit Column Selection
  - Don’t use the “select *” syntax
  - Selecting unused columns just increases execution time.
- `<CFTRANSACTION>` for performance
  - ColdFusion MX is READ-COMMITTED by default. (selects wait for insert/updates to complete)
  - If you don’t need up to the second results, use “READ-UNCOMMITTED” instead.
Tuning ColdFusion Applications

General Query Performance Tips (cont’d)

- Use **BLOCKFACTOR** to reduce network calls for many rows
- Use `<CFQUERYPARAM>` to leverage bind parameter & prepared statement speed for repeated calls
  - ESPECIALLY true for Type IV JDBC DSN’s (MSSQL, Oracle, IBM DB2, Sybase, Informix)
- Leverage the Database capabilities
  - Stored Procedures
  - Triggers
  - Defaults
- Choosing Database Drivers
Tuning ColdFusion Applications

- Query Caching
  - Two methods:
    - Store query result set in a persistent scope variable
    - Automatically by ColdFusion
  - Query Caching Considerations:
    - Reusability of result set
    - How often data changes in tables
    - Memory usage (large resultsets)
Tuning ColdFusion Applications

- Queries stored in persistent scope
  - Store result set in persistent scope variables
    - Session, Application, Server scopes
  - Typically in Application.cfm
  - Example:
    - Store query with 50 states
      - `<CFQUERY Name="Application.States"`
Tuning ColdFusion Applications

- Automatically Cached Queries
  - Two CFQUERY attributes
    - \textsc{cachedwithin}
    - \textsc{cachedafter}
  - All attributes and SQL must be the same
  - Dramatic performance increase
  - Use Query-of-Queries feature on cached queries
    - Best practice is to do one large query, cache it and use query-of-query instead of multiple calls to the db
Tuning ColdFusion Applications

External Calls

- Common external calls:
  - Database requests
  - Web Services, CFMAIL, CFPOP, CFHTTP, CFLDAP
  - CFX tags, COM objects, etc
- Can be long-executing and blocking calls
  - Can tie up worker (request) threads for long periods
  - Use timeout values wherever possible for safety
- If all worker threads block, new requests can queue up
  - Test and monitor external calls
  - Enable reasonable timeouts in external calls & 3rd party software
  - Can use Thread Dumps to troubleshoot
Tuning ColdFusion Applications

- Scoping
  - Explicitly scope your variables. Especially *dynamically evaluated variables*:
    - Helps the runtime by short-circuiting the variable lookup procedure
    - i.e.: USE `isDefined("form.#field1#")`
      INSTEAD OF `isDefined("#field1#")`

**GOOD**

```cfml
<cfloop list="#form.fieldnames#" index="field">
  <cfoutput>#field#: #form[field]#<br></cfoutput>
</cfloop>
```

**BAD**

```cfml
<cfloop list="#form.fieldnames#" index="field">
  <cfoutput>#field#: #evaluate(field)#<br></cfoutput>
</cfloop>
```

- Can result in dramatic performance impact, depending on code (in a loop, etc)
Tuning ColdFusion Applications

CFCACHE Tag -- Page Content caching

- Good tag: use wherever possible for dramatic overall site performance gain
- Page is “preprocessed”
  - All or part of page can be cached
    - CFCACHE can be in middle of page
  - Result stored in static HTML file to filesystem

- Usage guidelines:
  - Use where content delivered doesn’t change for some period
  - Specify timespan with max acceptable delay for change to take effect
  - Use action="ClientCache" for personalized pages that don’t otherwise change for some period of time
  - Don’t use in cases where you have large number of cached entries (filesystem seek time) use in-memory solution instead
Tuning ColdFusion Applications

CF_Accelerate In-Memory Caching Solution

- Stores page content in Application scope
- Cache entire page or partial page data
- Stores content in a tree of structures for faster seek times in very large caches
- Default is to cache unique data based on URL parameters
- Management interface for viewing & managing cached data

Usage
<cf_accelerate>
...page content....
</cf_accelerate>

- http://www.bpurcell.org/blog/index.cfm?mode=entry&ENTRY=963
- Code Example
CFFLUSH Tag (User Perception)

- Outputs partial page results
- Pages perceived as running faster
- Good for long running operations
- Can do “flush now” or flush every N bytes
- Examples:
  - `<CFFLUSH>`
  - `<CFFLUSH Interval="1000">`
Misc Coding Practices

- Reusing CFC instances by storing a reference in a shared scope.
  Application.mycomp=CreateObject("component","dir.componentname")

- List vs. Array
  Accessing items in an array is up to 10x faster than a list. May want to convert lists to an array and work with them and convert back if needed.
Tuning ColdFusion Settings

- ColdFusion MX settings
  - Simultaneous Requests
  - Template Cache Size
  - Trusted Cache
  - Client Variables
  - Whitespace Management
  - Datasource Settings
  - Logging long running requests
Tuning ColdFusion Settings

- Simultaneous Requests
  - Controls number of worker (request) threads created and able to service requests simultaneously
  - Single most important CF Admin setting
  - Under the hood, this sets the JRun “ActiveHandlerThreads”
  - No magic formula, BUT:
    - Optimal CF5 setting may be too low for CFMX
    - Generally, start tuning at **3 per CPU** (i.e.: 6 on 2-way or 12 on 4-way)
    - Too high can cause heavy context switching
    - Optimum Sim Req setting for your application:
      - Use a load test tool and the testing process (test, adjust setting, test, repeat)
    - CFMX 6.1 default is 8 (JRun 4.0 default is 25)
Tuning ColdFusion Settings

- Template Cache Size
  - Number of templates to keep in cache
  - In CF 5 it was memory size
  - If you have a large number of templates and set this large value (2,000-10,000+) you may see OutOfMemory errors
    - Permanent generation of JVM heap will go over default setting of -XX:MaxPermSize=128m
    - If this occurs increase -XX:MaxPermSize=192m or -XX:MaxPermSize=256m
Tuning ColdFusion Settings

- **Trusted Cache**
  - If “ON”, no check of template source file date/time
  - If “OFF”, will check and recompile templates if newly changed
  - Turn it “ON” for production
  - Enabling “Trusted Cache” Admin switch minimizes filesystem stat() calls
    - Includes Application.cfm and OnRequestEnd.cfm
  - Can **dramatically** increase performance under load
    - Especially on shared network file systems
  - No server restart required
    - You can deploy new files dynamically
Tuning ColdFusion Settings

- **Client Variable Settings**
  - Client variable backing store settings
    - Cookies are most scalable option
    - DB is next scalable option
    - Registry (default) is **least scalable** & not “farm-friendly”

- Review your `<CFApplication>` tag, only set `clientmanagement="yes"` if you are using client variables. If it is inadvertently set to true a client variable entry will be created for every user.
Tuning ColdFusion Settings

- Enable Whitespace Management
  - Trade-off between extra server work and download speed
  - Can be expensive on server side
    - As an alternative: HTTP compression
  - Test ON and OFF, measuring TTLB in expected production & customer scenarios
Data Source Settings

- Maintain DB connections should = ON unless you have a good reason not to (MS Access)
- Don’t limit DB connections unless you have a good reason
  - Usually need one connection per simultaneous request
  - If database server capability is stretched, limiting connections may help overall performance
- Use a Type IV JDBC driver!!
- Disable "BLOB" and "CLOB" if not using
Tuning Application Server Settings

- Application Server Settings
  - JVM
  - Heap Size
  - Garbage Collection
  - Threads a.k.a. simultaneous requests
  - Queuing
Tuning Application Server Settings

- JVM
  - Which one?
    - Sun
    - BEA JRockit
    - IBM
  - Test thoroughly with load tool
  - Never switch in production
- CFMX 6.1 ships with Sun 1.4.2-b28
- Try the latest from Sun while load testing 1.4.2_04
Tuning Application Server Settings

- **Heap Size**
  - `-Xms` and `-Xmx` JVM arguments
  - Needs to be appropriately large within constraints of available physical memory
  - Avoid extending into page file (swap)
  - On production server 50-75% physical RAM as rule of thumb, i.e. 512MB – 768MB for server with 1GB
  - Find optimal size and then set min and max to the same value.
  - If app is small or using multiple instances, no need to set too high.
  - If too small, OutOfMemory errors
  - Setting `-Xms` and `-Xmx` to the same value increases predictability by removing the most important sizing decision from the virtual machine
Tuning Application Server Settings

- Garbage Collection
  - Complex but important topic to performance
  - See JVM vendor docs
  - Each JVM offers numerous GC options
    - `-XX:+UseConcMarkSweepGC`
    - `-XX:+UseParallelGC`
  - Throughput GC vs. low pause GC
  - Additional JVM args control characteristics of heap and garbage collection.
    - `-XX:+UseAdaptiveSizePolicy`
    - `-XX:+AggressiveHeap`
      - Sun only
      - don’t use Xms/mx with AggressiveHeap
  - Good Summary on Tuning
    http://www.petefreitag.com/articles/gctuning/
Tuning Application Server Settings

- **Threads a.k.a. simultaneous requests**
  - All application servers provide a way to control this setting.
  - As discussed earlier, very important to performance
  - ColdFusion simultaneous requests map to JRun ActiveHandlerThreads
  - i.e. WebSphere’s web container queue settings

- **Queuing**
  - Application server specific settings
Monitoring Performance

- CFSTAT and Performance Counters
- JRun Metrics
- ColdFusion Debug Output
- ColdFusion MX & JRun logs
- JVM Verbose Heap Data
- Network Monitoring & CPU
Monitoring Performance

- CFSTAT and Performance Counters
  - /opt/coldfusionmx/bin/cfstat
  - Best window into what CFMX Server is doing
    - Not available on J2EE install
    - Other appservers- use native tools i.e. Tivoli Performance Viewer on WebSphere
Monitoring Performance

- CFSTAT and Performance Counters (cont’d)
- Enable CFSTAT in the ColdFusion Administrator under Debugging & Logging -> Debugging Settings

☑ Enable Performance Monitoring
Select this check box so the standard NT Performance Monitor application shows information about a running ColdFusion application server.

☑ Enable CFSTAT
The cfstat command-line utility provides real-time performance metrics for ColdFusion. Using a socket connection to obtain metric data, cfstat displays the information that ColdFusion writes to System Monitor without actually using the System Monitor application.
Monitoring Performance

- CFSTAT and Performance Counters (cont’d)
- Provides realtime statistics on how the server is running
- What information does it provide
  - Pg/Sec
  - Db/Sec
  - CP/Sec – Cache Pop/sec
  - Requests queued
  - Requests Running

```
D:\CFusionMX\bin>cfstat 2

Pg/Sec  DB/Sec  CP/Sec  Req's Q'ed  Req's Run'g  Req's TO'ed  AvgQ  AvgReq  AvgDB  Bytes In/Sec  Bytes Out/Sec
1  1  3  3  -1  -1  0  8  0  0  16853  4  161  58194
1  1  13  13  -1  -1  0  8  0  0  20571  0  225  81471
1  0  13  13  -1  -1  0  8  0  0  20571  0  0  0
0  1  0  13  -1  -1  0  8  0  0  20571  0  0  0
0  1  5  13  -1  -1  0  8  0  0  17601  5  100  36196
0  1  0  13  -1  -1  0  0  0  0  17601  5  0  0
1  1  10  13  -1  -1  0  8  0  0  15411  10  200  72389
```
Monitoring Performance

- JRun metrics works on JRun and ColdFusion MX standalone. The standalone configuration is a little bit different for details see http://www.bpurcell.org/blog/index.cfm?mode=entry&entry=991

- Reasons to use Metrics
  - When server is hanging regularly
  - Server is responding slowly
  - Monitoring During load tests
Monitoring Performance

- JRun Metrics (cont’d)
  - By customizing JRun metrics we can provide a lot of information about how the server is performing and the number of requests it is processing
  - Listing of additional metrics settings
    AvgQueueTime, AvgReqTime, busyTh, bytesIn, bytesOut, delayMs, delayRq, delayTh, droppedRq, freeMemory, handledMs, handledRq, idleTh, listenTh, sessions, sessionsInMem, totalMemory, totalTh
  - Customizing Metrics
    http://www.bpurcell.org/blog/index.cfm?mode=entry&entry=991
Monitoring Performance

- CFMX Debugging Performance Output:

<table>
<thead>
<tr>
<th>Execution Time</th>
<th>File Path</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>(62ms)</td>
<td>C:\Inetpub\wwwroot\allaire\spectra\examples\i-build\Application.cfm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\application\cfa_applicationinitialize.cfm</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\utils\cfa_globalsettings.cfm</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\coapi\utils\cfa_datasourcegetdbms.cfm</td>
<td>240</td>
</tr>
<tr>
<td>(1219ms)</td>
<td>C:\Inetpub\wwwroot\allaire\spectra\examples\i-build\index.cfm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\sitemodeling\cfa_page.cfm</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\i-build\ibuildformatting.cfm</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\publishing\cfa_container.cfm</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\i-build\ibuildsearchbox.cfm</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\i-build\ibuildtopicbox.cfm</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier0\metadata\cfa_metadatatypeget.cfm</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier0\objectstore\cfa_contentobjectget.cfm</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\coapi\utils\cfa_entityconstants.cfm</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier0\objectstore\cfa_contentobjectcacheread.cfm</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\publishing\cfa_container.cfm</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>C:\Program Files\allaire\spectra\customtags\system\tier1\publishing\cfa_container.cfm</td>
<td>49</td>
</tr>
<tr>
<td>(0 ms)</td>
<td>STARTUP, PARSING, COMPILING, LOADING, &amp; SHUTDOWN</td>
<td></td>
</tr>
<tr>
<td>(1281 ms)</td>
<td>TOTAL EXECUTION TIME</td>
<td></td>
</tr>
<tr>
<td>red = over 100 ms execution time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Always turn off debugging when load testing and in production, just masking IP’s will not work debugging still runs output is just masked
Monitoring Performance

- Following the Logs during Development & Load Testing
- What logs are available
  - JRun
    - \{jrun-root\}/logs
  - ColdFusion MX Standalone
    - \{cfmx\}/logs
    - \{cfmx\}/runtime/logs
- Options to Configure in ColdFusion MX admin
  - Logging long running pages
  - Timing out long running requests
    (does not affect database calls and third party code…COM, Java…etc)
Monitoring Performance

- JVMVerbose Output
  - Provides information on memory management within the JVM
  - Provides information on time spent doing GC (garbage collection)
  - When to use verbose GC output
    - When you are getting OutOfMemory Exceptions
    - When you see the server suddenly stop responding for long periods of time then start responding again.
Monitoring Performance

- JVM Verbose Output
  - How do you enable JVM Verbose GC output
    - In the `{jrun-root}/bin/jvm.config` (or) `{cfroot}/runtime/bin/jvm.config`
    - Add the following arguments to the `java.args`
      - `-XX:+PrintGCDetails` `-XX:+PrintGCTimeStamps` `-XX:+PrintHeapAtGC`
    - For full details see Tuning Garbage Collection at
      [http://java.sun.com/docs/hotspot/gc1.4.2/](http://java.sun.com/docs/hotspot/gc1.4.2/)
Monitoring Performance

- JVM Verbose Output
- Sample JVM Verbose GC output

Heap

PSYoungGen total 7616K, used 7611K [0x10010000, 0x108b0000, 0x11c80000)
  eden space 6400K, 99% used [0x10010000,0x1064fff0,0x10650000)
  from space 1216K, 99% used [0x10780000,0x108aed50,0x108b0000)
  to space 1216K, 0% used [0x10650000,0x10650000,0x10780000)
PSOldGen total 5312K, used 2138K [0x11c80000, 0x121b0000, 0x20010000)
  object space 5312K, 40% used [0x11c80000,0x11e96b60,0x121b0000)
PSPermGen total 16384K, used 9674K [0x20010000, 0x21010000, 0x28010000)
  object space 16384K, 59% used [0x20010000,0x20982958,0x21010000)
16.377: [GC 9750K->3747K(14144K), 0.0197704 secs]

- Three areas (aka generations) where memory is stored in the JVM
  - YoungGeneration (or Eden)
  - OldGeneration (or Tenured)
  - Permanent Generation
Monitoring Performance

- Generations of the JVM
Monitoring Performance

- Network Monitoring
  - Netstat
  - Third party tools

- CPU
  - During load tests you want to saturate the CPU to near 100% if you cannot the bottleneck is the number of threads or third party calls
  - Use top or prstat on Unix
  - Use Task Manager or Perfmon on Windows
Conclusion

- Measure baseline
- Identify Bottlenecks, e.g. code, settings
- Change a single piece of code or setting
- Measure again
- make another change
- Measure again
  - and change and measure and change and measure....
  - etc.
- Go faster
Thank you.