Scaling Applications (optional)

Objectives

After completing this unit, you should be able to:

- Configure the ColdFusion Server for optimal performance
- Cache query data in persistent memory
- Cache sections of generated page content in persistent memory
- Cache entire page results to disk
- Develop processes that execute asynchronously
- Monitor the performance of your server in real-time

Topics

In this unit, you will learn about the following topics:

- Configuring for Scalability
- Understanding and Identifying Scalability Bottlenecks
- Using Asynchronous Processes to Improve Performance
- Caching Recordsets
- Caching Content with <cfcache>

Configuring for Scalability

- As a J2EE-based application, ColdFusion relies on the stability and performance of several products from third-parties
- Configuring your Java Virtual Machine (JVM) which is responsible for processing most requests, is critical
- Many of the deployment problems encountered by ColdFusion Developers and administrators center around memory allocation
  - The amount of memory that a server can access is limited
  - You can configure how your memory is organized
  - Garbage collection can affect scalability
- Maximizing the performance of your server is also highly dependent on choices that you make in the ColdFusion Administrator regarding the handling of simultaneous requests
- Mechanisms that need to be configured properly lest they adversely impact scalability include the following:
  - Datasources
  - Charting
  - Flex form compilation
  - Verity search engine
  - Solr search engine

Understanding JVM Memory Limitations

- The default installation of ColdFusion allocates a maximum of 512MB to the ColdFusion server
- ColdFusion 32-bit on Windows is limited to a maximum of 1.2 GB
- ColdFusion 64-bit can address a virtually unlimited memory space
- SUN recommends that you not allocate more than 2GB to any instance due to diminishing returns related to garbage collection
- You can encounter memory issues if your applications perform any of the following:
  - Self-generation of CFM files (common in many content management systems)
  - Aggressively use in-memory caching of queries and output
  - If any of the third-party CFX or Java components that comprise your application have a memory leak where used address space is never returned to the heap
  - Instantiation of a large default session scope while answering requests from a large number of concurrent users and setting session timeout to a large number

Modifying JVM Settings

- Edit the jvm.config file using a text editor
  - Single server: [cfinstall path]\runtime\bin\jvm.config
  - Multi server: [jrun install path]\bin
- ColdFusion single-server allows you to modify the jvm configuration through the ColdFusion Administrator
- Any error in the format of the JVM arguments will prevent the ColdFusion service from starting successfully

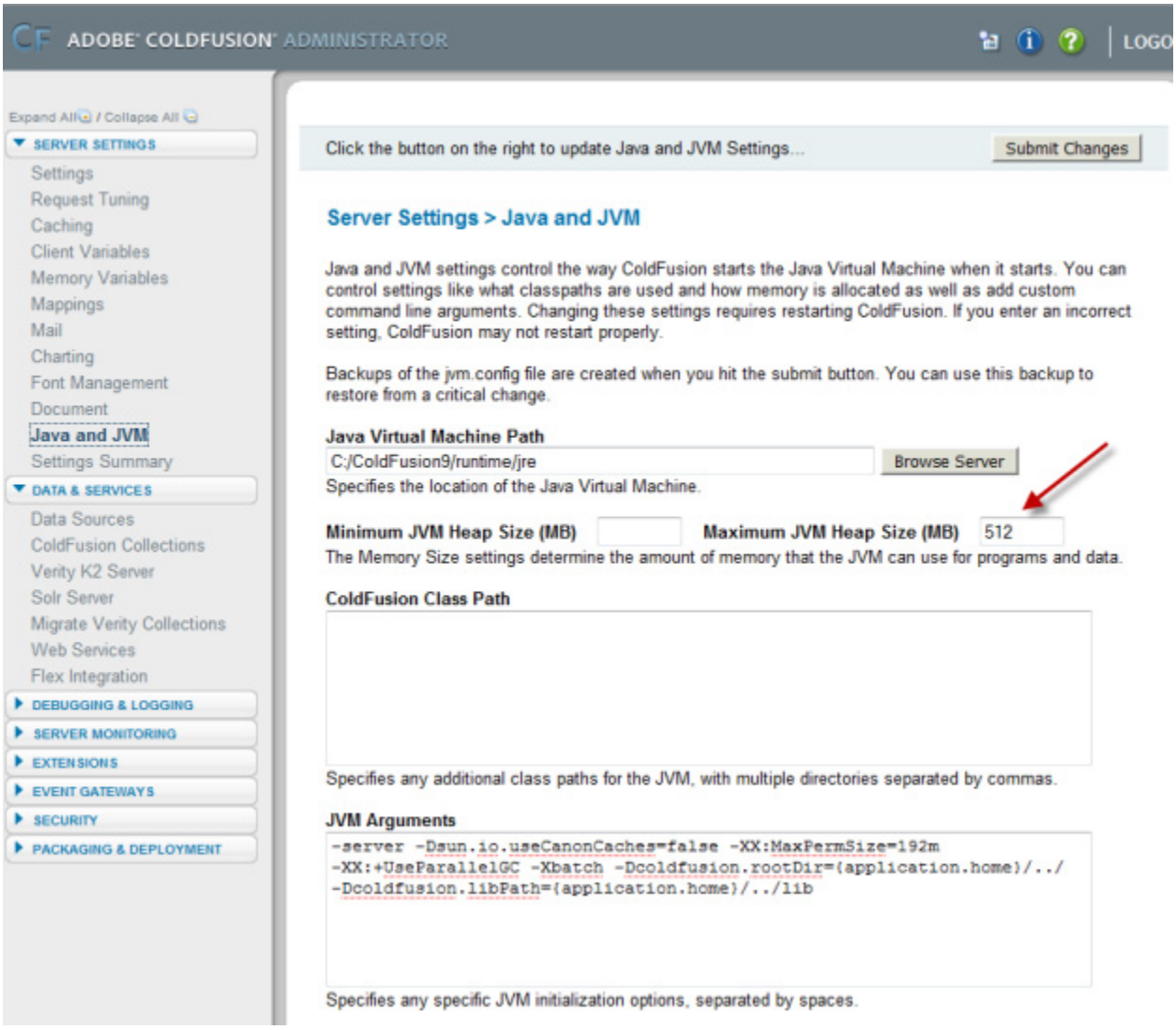


Figure 1: Modifying JVM settings through the ColdFusion Administrator. Note the default max memory size of 512MB

Working with JVM Argument Syntax

- Boolean options are turned on with -XX:+<option> and turned off with -XX:-<option>.

- Numeric options are set with -XX:<option>=<number>. Numbers can include 'm' or 'M' for megabytes, 'k' or 'K' for kilobytes, and 'g' or 'G' for gigabytes (for example, 32k is the same as 32768)
- String options are set with -XX:<option>=<string>, are usually used to specify a file, a path, or a list of commands

Note: With each succeeding iteration of its JVM, Sun strives for ergonomic improvements, so that manual changes to the jvm arguments can be minimized

Standard JVM Settings

You have probably noticed the following switches in your java.config file:

Setting	Example	Description
-server	-server	Loads the Java Hotspot server VM (as opposed to -client which loads the client VM)
-D*property=value*	-Dsun.io.useCanonCaches=false	Sets a system property value
-Xmx*MemSize*	-Xmx512m	Specify the maximum size, in bytes, of the memory allocation pool. This value must a multiple of 1024 greater than 2MB. Append the letter k or K to indicate kilobytes, or m or M to indicate megabytes. The default value is 64MB.
-Xms*MemSize*	-Xms512m	Specify the initial size, in bytes, of the memory allocation pool. This value must be a multiple of 1024 greater than 1MB
-XX:*MaxPermSize*	-XX:MaxPermSize=64m	Maximum size of the Permanent Generation. 64-bit VM's are 30% larger
-XX:*PermSize*	-XX:Permsize=64m	Initial size of the permanent generation

Modifying Request Settings

- The following settings in the CF Administrator signficantly affect scalability and performance.
- Properly configuring your CF instance through the ColdFusion Administrator can return performance gains of 30% or more
- Consult your ColdFusion Administrator guide for more details

- Request Limits
- Maximum number of running JRun threads
- Request Timeout
- Maximum size of post data
- Enable Whitespace Management
- Enable Global Script Protection
- Maximum number of cached templates
- Disable CFC Type Check
- Trusted Cache
- Save Class Files
- Cache Web Server Paths
- Maxmum number of cached queries
- Memory variable timeouts
- Client variable configuration
- Enable in-memory file system
- Application.cfc/ Application.cfm lookup order

Walkthrough 1: Configuring for Scalability

In this walkthrough, you will perform the following tasks:

- Increase the amount of memory available to your ColdFusion server instance
- Review ColdFusion Administrator settings that impact scalability with your instructor

Steps

Allocate more memory to your CF Server

1. Open the ColdFusion Administrator in your web browser using the following URL:

```
http://localhost:8500/cfide/administrator/
```

2. Click on Server Settings > Java and JVM
3. Change the Minimum Heap Size to 900 and the Maximum heap size to 1200
4. In the JVM Arguments section, set -XX:MaxPermsize equal to 256m
5. Click Submit Changes
6. Restart your ColdFusion server
7. Return to the ColdFusion Administrator

Review and Modify Server Settings

8. Click Server Settings > Settings
9. Turn on the checkbox to Timeout Requests after 30 seconds
10. Turn on the checkbox to Enable WhiteSpace Management
11. Turn on the checkbox to Disable CFC Type Check
12. Turn on the checkbox to Enable In-Memory File System
13. Turn on the checkbox to Enable Global Script Protection
14. Note the other options which affect memory consumption
15. Click Submit Changes
16. Click on Server Settings > Request Tuning
17. How can you determine the optimal value for Maximum number of simultaneous Template requests?
18. Set the Maximum number of simultaneous template requests to 8
19. Click Submit Changes
20. Click Server Settings > Caching
21. Review the options with your instructor
22. Leave your browser open for the next walkthrough

Understanding and Identifying Scalability Bottlenecks

- Scalability cosiderations involve how well your application supports multiple users simultaneously accessing your system.
- An optimally scalable application will provide a consistently acceptable response time for each concurrent user of your system.
- No system is infinitely scalable, however, the performance of your application should degrade in proportion to the number of concurrent users.
- Scalability can be measured by performing load testing whereby an automated program simulates concurrent users by running use-case scripts against your application.
- Items affecting scalability include, but are not limited to, the following:
  - ColdFusion server configuration
  - Server hardware
  - Database performance
  - Server network infrastructure
  - Source code efficiency under load

Minimizing the amount of work performed

- Solving scalability issues involves doing as little work at runtime as possible by minimizing the amount of work that needs to occur on any user-driven page request.
- Achieving this may involve doing additional work either before or after the initiated request.
- Meeting your scalability goals typically involves using one or more of the the following techniques:
  - Caching query data in server memory, thereby reducing calls to the database
  - Caching output results in either server memory or on disk, reducing calls to business and display logic.
  - Executing long running processes asynchronously, or scheduling them to run during off-peak hours.
- The following tags, by their nature, are process-intensive due to memory consumption, disk i/o, or network i/o. Whenever possible, try to minimize the number of times they execute through caching:

Tag	Description
<cfdocument>	PDF generation, may involve external HTTP calls

Tag	Description
<cfpdf>	PDF manipulation, usually involving disk I/O
<cflock>	Single-threads execution to avoid concurrency problems
<cfdirectory>	Reads directory information from the server or shared volume
<cfimage>	Reads and manipulates image data
<cfftp>	File Transfer (FTP) operations
<cfhttp>	Retrieves data from an external server via HTTP
<cffile>	File I/O functions
<cffeed>	Retrieves and parses external RSS/ ATOM feeds
<cfform format=”flash”>	Flash form generation
<cfobject>/<cfinvoke>	Execute an external web service / load a local CFC
<cfchart>	Dynamically creates a JPG or Flash-based chart
<cfcollection>	Creates a verity collection
<cfindex>	Indexes a verity collection
<cfpresentation>	Dynamically assembles a Flash-based multimedia presentation
<cfspreadsheet>	Manages Excel Spreadsheet files

Identifying slow performers

- The first step towards producing a highly scalable application is to identify areas within your codebase that perform slowly as compared to the rest of the application.
- You also need to locate any code that produces unexpected results when executed simultaneously by multiple users. In order to isolate these issues.
- Consider the following:
  - Metrics are available from ColdFusion's debug output and logging that can help you identify poor performing pages.

- You can use the `<cftimer>` tag and `getTickcount()` function to isolate the performance of specific lines of code.
- Using load testing tools in conjunction with ColdFusion's server monitor can help you understand how your application and infrastructure will perform when handling multiple simultaneous requests.
- Load test results vary depending on the system configuration being tested, e.g. a system with 512MB of RAM allocated to ColdFusion may perform differently than one with a 2GB allocation.

Leveraging Debug Output

- ColdFusion's debug output is your first line of defense for identifying slow-running pages, components, and queries.
- When working on a development server, enable debugging through the ColdFusion Administrator and turn on the checkboxes for each of the following:
  - Report Execution Times: This option will display the execution time for each module called as part of a ColdFusion request. This feature can help isolate poor performing modules within a request.
  - Database Activity: This option displays all queries and stored procedures that executed during the page request, along with their execution time and the number of rows returned. This option will isolate poor performing queries as well as those returning larger than expected recordsets. You may also find instances where the same query is repeating multiple times.
  - Timer Information: This option must be enabled in order to use the `<cftimer>` tag to diagnose performance issues with specific lines of code.
- In the figure below, we see page execution time for walkthrough 6-5 (Microsoft Excel spreadsheet generation).
  - Note that the execution times for each CFC method, custom tag, and included file are listed separately.
  - You typically need to run the page twice in quick succession in order to get a more accurate analysis of run-times due to ColdFusion's template caching mechanisms
  - Operations that are heavily dependent on disk i/o or make external http requests to servers will be slow as compared to pages that leverage only <cfquery> and <cfoutput>

Debugging Information

ColdFusion Server Developer	9,0,0,251028
Template	/acfd9/walksolution/unit6/walk6/walk6-5.cfm
Time Stamp	28-Dec-09 03:23 PM
Locale	English (US)
User Agent	Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 GTB6 (.NET CLR 3.5.30729)
Remote IP	0:0:0:0:0:0:0:1
Host Name	0:0:0:0:0:0:0:1

Execution Time

Total Time	Avg Time	Count	Template
4558 ms	4558 ms	1	C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\walk6-5.cfm
203 ms	102 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | execute()] from C:\ColdFusion9\CustomTags\com\adobe\coldfus
198 ms	198 ms	1	CFC[C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\components\crime.cfc | init(ABQIAAAAPnOo-9KzoTHfxfMaLnmE0RT2yX
190 ms	190 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | send()] from C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.c
123 ms	123 ms	1	CFC[C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\components\crime.cfc | getCrimeStats(01/01/2008, 01/01/2009)] from C:\Cold
91 ms	91 ms	1	CFC[C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\components\crime.cfc | getCrimeDetails()] from C:\ColdFusion9\wwwroot\ac
17 ms	17 ms	1	CFC[C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\Application.cfc | onRequestStart(/acfd9/walksolution/unit6/walk6/walk6-5.cfm
2 ms	2 ms	1	C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\customtags\crimereportingform.cfm
1 ms	1 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | addParam(name = key, value = ABQIAAAAPnOo-9KzoTHfxfMaLnmE0R
1 ms	1 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addParam(cfsqltype = cf_sql_date, value = 01/01/2008, type = in)
1 ms	1 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addParam(cfsqltype = cf_sql_float, value = -77.036777, type = in)] f
1 ms	1 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addProcResult(name = q, resultset = 1)] from C:\ColdFusion9\C
0 ms	0 ms	1	C:\ColdFusion9\wwwroot\acfd9\walksolution\unit6\walk6\customtags\portlet.cfm
0 ms	0 ms	1	C:\ColdFusion9\wwwroot\acfd9\webportal\login\login_include.cfm
0 ms	0 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | INIT(method = get, url = http://maps.google.com/maps/geo)] from C:\Col
0 ms	0 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | addParam(name = output, value = json, type = url)] from C:\ColdFusion
0 ms	0 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | addParam(name = q, value = 1400 16th Street NW Washington,DC, type =
0 ms	0 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\http.cfc | addParam(name = sensor, value = false, type = url)] from C:\ColdFusion
0 ms	0 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | INIT()] from C:\ColdFusion9\CustomTags\com\adobe\coldfusion
0 ms	0 ms	1	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addParam(cfsqltype = cf_sql_date, value = 01/01/2009, type = in)
0 ms	0 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addParam(cfsqltype = cf_sql_float, value = 0.1, type = in)] from C:\
0 ms	0 ms	2	CFC[C:\ColdFusion9\CustomTags\com\adobe\coldfusion\storedproc.cfc | addParam(cfsqltype = cf_sql_float, value = 38.909085, type = in)]
9 ms			*STARTUP, PARSING, COMPILING, LOADING, & SHUTDOWN*
4584 ms			*TOTAL EXECUTION TIME*

Figure 2: Debug output from the dynamic spreadsheet creation in walkthrough 6-5.

Note: As a general rule, in a development environment, you should strive to engineer your pages so that they execute in no more than 100ms. Do not enable debugging on a production server as it significantly degrades performance.

Using <cftimer>

- The `<cftimer>` tag allows you to measure the execution time for arbitrary areas of a page.

- In order to use this tag, debug output must be enabled and the timer information checkbox should be checked.
  - The `label` attribute is a simple text string that is used to differentiate between multiple `<cftimer>` blocks.
  - The `type` attribute accepts the following four values:

Type	Description
inline	displays timing information inline with your generated html
outline	Similar to inline, but displays a line around the output produced by the timed code. The browser must support the <fieldset> tag for the line to be visible
comment	Displays timing information output within HTML comment tags
debug	Displays timing information in the debug output

- In the following example, `<cftimer>` is used to ascertain how long it takes to read, modify, and write a spreadsheet file.

```
<cftimer label="output block" type="debug">
   <cfspreadsheet action=”read” ... >

   ...

   <cfspreadsheet action=”write” ....>
</cftimer>
```

Load Testing

- In order to completely understand the scalability limitations of your application, you must test it using multiple clients.
- Load testing tools enable you to generate multiple simultaneous requests.
  - Test using specific representative paths through the application that are representative of typical use cases.
  - As the testing progresses, gradually increase the number of "virtual users," measuring response times and/or throughput.
  - You continue testing until the server breaks or an intolerable response time is reached.

Note: Keep in mind that virtual users don't pause while executing the use-case scripts and they have no explicit bandwidth limitations.

- Packages and services are available from third party vendors to help you understand the limits of your application and perform capacity planning such as:
  - Borland Silkperformer - http://www.borland.com/us/products/silk/silkperformer/index.html
  - Empirix - http://www.empirix.com/products-services/w-testing.asp
  - Apache JMeter http://jakarta.apache.org/jmeter/

Using the ColdFusion Server Monitor

- ColdFusion includes an Adobe Flex-based GUI for monitoring server performance.
  - Quickly identify common problems such as memory leaks, poorly performing pages and slow queries.
  - Configure alerts to take corrective action and/or send out notifications when specific problem thresholds are reached.
  - Take a "snapshot" of performance that outputs current memory status, load, and detailed information about currently executing threads.

Starting the monitoring process

- Launch the server monitor

1. Launch the ColdFusion Administrator
2. Select Server Monitoring > Server Monitor > Launch Server Monitor

http://[server]/CFIDE/administrator/monitor/launch-monitor.cfm

- In the executive dashboard, you will see the control bar:

Figure 3: The ColdFusion Monitor Control Bar

- In order to start capturing statistics, you will need to click on one or more of the following buttons:
  - Start Monitoring
    Clicking this button starts the process of gathering information about requests and is used to generate critical reports such as slowest requests, active sessions, and cumulative server usage.

- Start Profiling
  Clicking this button gathers information about how long it takes the server to process queries and requests – information that is critical in locating bottlenecks within your application. When profiling is activated you can drill into poorly performing requests or those consuming a lot of memory in order to determine why your request is misbehaving.
- Start Memory Tracking
  This feature causes ColdFusion to track information about overall memory usage, including how much is being consumed by queries, sessions, and shared memory scopes. Note that this process is very resource intensive will negatively impact performance.

The Executive Dashboard

- The Executive Dashboard, as depicted below, displays the following information about your server:
  - Average Response Time
  - Requests per second
  - Slowest Active Requests
  - Alerts
  - Last Error
  - Request / Query / JVM Reports
- Clicking on any of the reports brings up the same detailed information that you will find by clicking on the report via the Statistics tab.

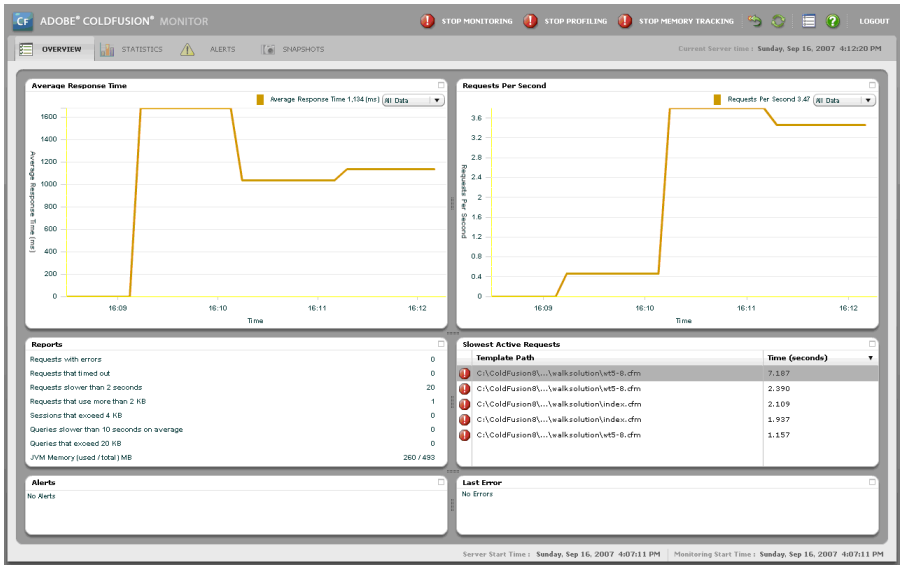


Figure 4: The Server Monitor Executive Dashboard

Monitoring Memory Usage

- The Memory Usage Summary report illustrated below is of particular interest when attempting to diagnose server instability.
- Since ColdFusion can only address a limited amount of memory, as defined in the jvm.config file, any process that consumes an inordinate amount of RAM for an extended period of time could cause the server to start processing requests inefficiently.
- You are strongly encouraged to run 64-bit ColdFusion in your production environment as it performs faster and can address far more memory than its 32-bit counterpart.
- You should take care to eliminate any code that generates unusually large, persistent memory requests before taking your application live.
- Do not run Query of Queries functionality on a large dataset on every request.
- Clicking on the Run GC button forces a garbage collection event whereby the JVM attempts to reclaim memory spaces that are no longer relevant to program execution and returns it to the general memory pool for re-allocation.

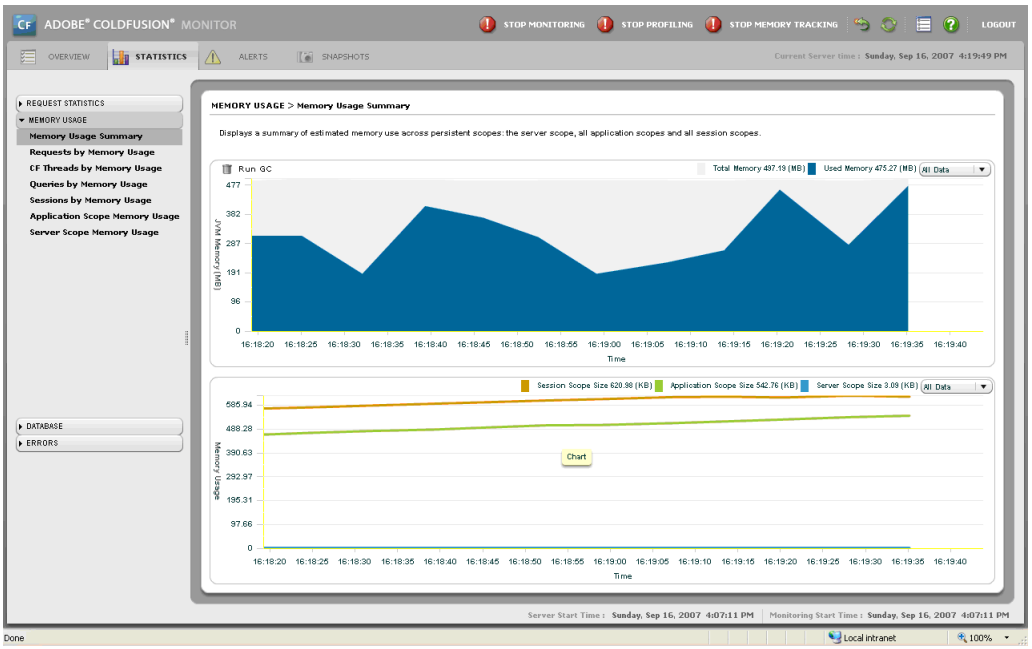
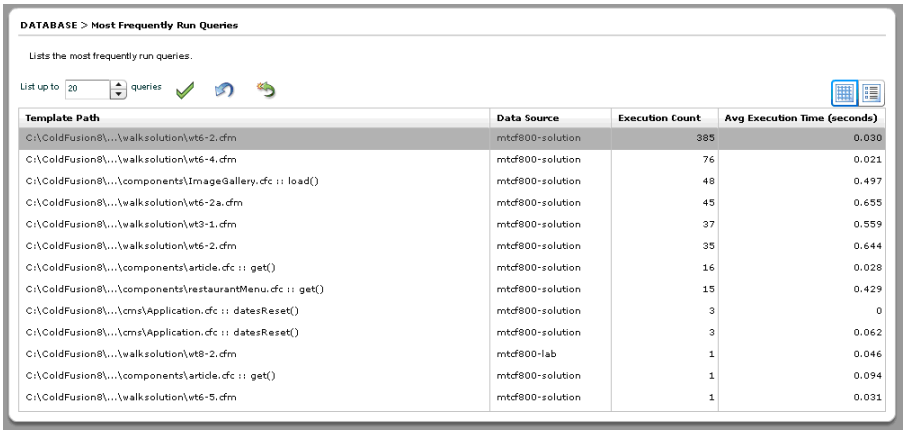


Figure 5: ColdFusion Memory Monitoring

Determining Most Frequently Run Queries

- As indicated below, the Most Frequently Run Queries identifies pages where the most number of queries are being executed.
- While it does not actually identify the specific SQL that is being executed, it does give you the line number of the specific ColdFusion file where the data request can be found.



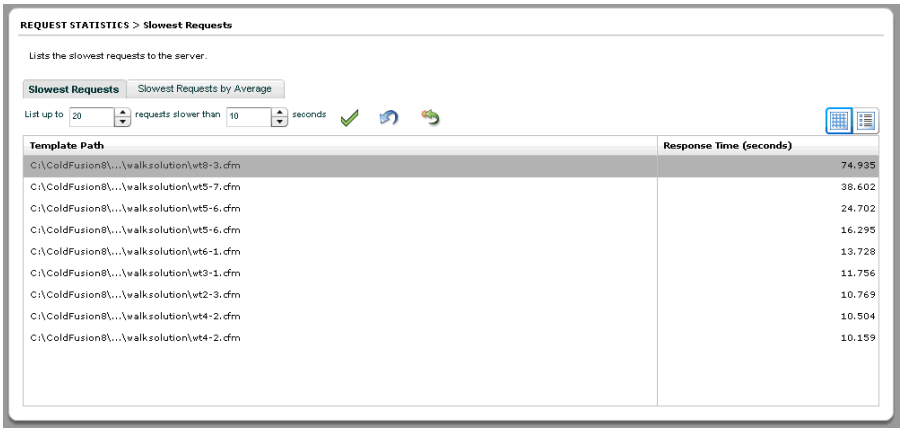
Template Path	Data Source	Execution Count	Avg Execution Time (seconds)
C:\ColdFusion8\...\walksolution\wt6-2.cfm	mtcf800-solution	385	0.030
C:\ColdFusion8\...\walksolution\wt6-4.cfm	mtcf800-solution	76	0.021
C:\ColdFusion8\...\components\ImageGallery.cfc :: load()	mtcf800-solution	48	0.497
C:\ColdFusion8\...\walksolution\wt6-2a.cfm	mtcf800-solution	45	0.655
C:\ColdFusion8\...\walksolution\wt3-1.cfm	mtcf800-solution	37	0.559
C:\ColdFusion8\...\walksolution\wt6-2.cfm	mtcf800-solution	35	0.644
C:\ColdFusion8\...\components\article.cfc :: get()	mtcf800-solution	16	0.028
C:\ColdFusion8\...\components\restaurantMenu.cfc :: get()	mtcf800-solution	15	0.429
C:\ColdFusion8\...\cms\Application.cfc :: datesReset()	mtcf800-solution	3	0
C:\ColdFusion8\...\cms\Application.cfc :: datesReset()	mtcf800-solution	3	0.062
C:\ColdFusion8\...\walksolution\wt8-2.cfm	mtcf800-lab	1	0.046
C:\ColdFusion8\...\components\article.cfc :: get()	mtcf800-solution	1	0.094
C:\ColdFusion8\...\walksolution\wt6-5.cfm	mtcf800-solution	1	0.031

Figure 6: The Most Frequently Run Queries Report

- Consider caching queries that have high execution counts and high execution times.
- Use the cachedwithin/cachedafter attributes, place problematic queries in the application or server memory space, or use <cfcache> to cache slowly executing queries in server memory.

Identifying Slow Requests

- The Slowest Requests report, as indicated below, displays long-running requests in your application.



Template Path	Response Time (seconds)
C:\ColdFusion8\...\walksolution\wt8-3.cfm	74.935
C:\ColdFusion8\...\walksolution\wt5-7.cfm	38.602
C:\ColdFusion8\...\walksolution\wt5-6.cfm	24.702
C:\ColdFusion8\...\walksolution\wt5-6.cfm	16.295
C:\ColdFusion8\...\walksolution\wt6-1.cfm	13.728
C:\ColdFusion8\...\walksolution\wt3-1.cfm	11.756
C:\ColdFusion8\...\walksolution\wt2-3.cfm	10.769
C:\ColdFusion8\...\walksolution\wt4-2.cfm	10.504
C:\ColdFusion8\...\walksolution\wt4-2.cfm	10.159

Figure 7: The Server Monitor can identify slow running requests

- Pages that are showing high response times generally indicate that a bottleneck exists within your application.
- Once you have identified the poorly performing pages, you may be able to compensate for their poor performance by implementing aggressive caching or redesigning your business logic to execute more efficiently.
- Execute your slow performing pages with debug output turned on in order to identity why the page is performing poorly.

Walkthrough 2: Identifying Bottlenecks

In this walkthrough, you will perform the following tasks:

- Analyze the performance of a page by using ColdFusion's debug output.
- Monitor server performance using the server monitor.

Steps

1. Return to your web browser containing the ColdFusion Administrator.

Start the Server Monitor

2. Click on Server Monitoring > Server Monitor
3. Click on Launch Server Monitor
4. Click on Start Monitoring
5. Click on Start Profiling
6. Click on Start Memory Tracking
7. Click on the Settings button, located in the top-right corner of the application
8. On the General tab, set each of the entries to 5 seconds
9. Click OK

Run the load testing script

10. In ColdFusion Builder, open the file /walk/unit8/walk2/walk8-2.cfm and review the contents with your instructor. You will cover `<CFTHREAD>` in more detail later in this unit.
11. Return to the ColdFusion Administrator and select Server Settings > Request Tuning
12. Enter 30 as the Maximum number of threads available for CFTHREAD. This will enable you to run all of the walkthrough examples concurrently.
13. Click Submit Changes
14. Return to walk8-2.cfm in ColdFusion Builder
15. Browse the file. As it is executing, toggle your browser to the server monitor and view the performance of the files that you created during this course.
16. Click on the Statistics tab
17. Select each of the available reports
18. Identify the requests that are putting the biggest load on the server by selecting Request Statistics > Cumulative Server Usage. Why do you think these files are causing bottlenecks? How would you address these issues?

Kill threads

19. Under the Request Statistics heading, select Active ColdFusion Threads

20. Abort the request KillServer1
21. Note how aborting this request impacts your server's memory utilization. by selecting Memory Usage > Memory Usage Summary. Your session memory usage continues to rise at approximately the same rate. Why?
22. Restart your ColdFusion server to quickly kill the rest of the threads.

Using Asynchronous Processes to Improve Performance

- Certain constructs are by their very nature resource intensive or time consuming, including, but not limited to, the following:
  - Full-text indexing with Verity (<cfindex>)
  - Operations that require a large amount of file i/o (<cffile>,<cfdirectory>)
  - Operations that rely on external server processing (<cfhttp>, <cfobject>)
  - Batch processes that involve a large number of iterations
- Consider architecting your code to execute asynchronously.
- Execute processes independently from the request that instantiates them.
- A user could receive an immediate notification that the request has started, however the page request does not need to wait for the async process to complete in order to continue with its execution.
- Precludes the async process from generating any output to the user. As such, results from running asynchronous processes are typically written to a database, log file, sent as an email, or sent to some other external resource.

Using <cfthread>

- ColdFusion allows you to fork your page execution into two or more simultaneously executing tasks.
- Since code segments executed through <cfthread> are executed asynchronously, a user need not wait for the forked code to complete executing in order for the parent page to complete its execution.
- Typically, kicking off threads can improve perceived performance for long-running requests that can execute in the background and do not necessarily need to send visual output to the user.
- The following tasks generally make excellent candidates for spawning as new threads:
  - Batch file processing, such as <cfzip>
  - Indexing of verity collections - <cfindex>
  - Creation of external files that are cached to disk - <cfpresentation>, <cfdocument>, <cfpdf>, <cfspreadsheet>, <cfimage>
  - Database transactions that do not return values (e.g. delete, update)
  - Processes that rely on external servers
    - <cfhttp>

  - <cfftp>
  - <cffeed>
  - <cfdirectory> run across a network volume
- In addition, <cfthread> can be helpful when aggregating content from tasks that can execute in parallel, such as merging content from multiple simultaneous <cfhttp> or <cffeed> requests.

Creating a new thread

- Instantiate a new thread through the <cfthread action="run"> tag.

```
<cfthread
  [required]
  name="thread name"
  [optional]
  action="run"
  priority="NORMAL|HIGH|LOW"
  zero or more application-specific attributes>

 [Code to execute asynchronously]

</cfthread>
```

- In the following example a verity collection is rebuilt. Normally the user would have to wait several seconds for the process to complete, however, by spawning it as a separate thread, the perceived execution time is zero miliseconds:

```
<cfthread action="run"
          name="rebuildVerityIndex" >

   <cftry>
     <cfcollection action="delete"
                   collection="acfd9" />

     <cfcatch />
   </cftry>

   <cfcollection
      action="create"
      collection="acfd9"
      path="c:\veritycollections" />
```

```
    <cfindex
      collection="acfd9"
      action="update”
      recurse="true"
      type="path"
      key="#expandpath('.')#"
      urlpath="#application.settings.basehref#">


</cfthread>
```

Additional <cfthread> Features

- <cfthread> can perform the following functions:
  - Spawn a new thread
  - Temporarily stop execution of a thread
  - Join code to an existing thread
  - Terminate a running thread
- You can restrict the number of simultaneous requests that ColdFusion can process through the ColdFusion administrator.
  - Ensures that each of your spawned processes receives enough attention from the CPU to finish processing in a timely fashion.
- In the event that you spawn code that does not complete, such as executing an infinite loop, accidentally programming a deadlock condition, or waiting forever for a <cfhttp> request to access a non-responsive site, note that the thread will, in fact, continue to execute.
- You can identify poorly performing threads through the ColdFusion Server Monitor.

Variable Scoping

- Inside threads there are three distinct variable scopes that you can reference.
  - The Thread-local scope applies to any variable defined inside the <cfthread> block without explicitly specifying a scope. It cannot be accessed or modified by outside of the <cfthread> block.
  - The Thread scope is available to the invoking page as well as all other threads spawned from the same CFM file. You define these variables by prefixing them with either your thread's name or the keyword thread.
  - The Attributes scope refers to any variables that were passed in through the <cfthread> tag invocation.
- All threads in a page share the variables, application, and server scopes.

Variable Scope Precedence

- If you fail to prefix your variables within a thread, the following scope-precedence applies:
  1. Local variables declared within the same UDF's in a thread
  2. Thread-local variables
  3. Attributes
  4. Variables
- In the following example, two entries are written to the log. The first entry is "Steve" and the second is "Dave".

```
<cfset var1 = "Steve">

<cfthread name="foo" action="run">
 <cfset foo.var1 = "Dave">
 <cflog text="#var1#" file="acfd9Solution">

 <cflog text="#thread.var1#"
        file="acfd9Solution">
</cfthread>
```

Thread Metadata

- The thread scope contains a number of variables that can be useful in determining the status of a thread.

Variable	Description
Thread.elapsedtime	The number of seconds that the thread has been executing.
Thread.error	A structure containing information about any error that occurs within the thread block
Thread.Name	The name of the thread, as specified in the <cfthread> tag
Thread.Output	Used to output content generated by a thread
Thread.priority	The thread processing priority, as defined in the <cfthread> invocation.
Thread.starttime	The time that the thread was spawned.
Thread.status	The current status of the thread. Returns one of the following: • Not_Started • Running • Terminated • Completed • Waiting

- In the following example, an output loop is executed asynchronously. The parent page is put into a wait-state until the thread has completed processing

```
<cfthread name="mythread" action="run">
 <cfloop from="1" to="100000" index="i">
  <cfoutput>t:#i#<br></cfoutput>
 </cfloop>
</cfthread>

<cfloop
condition="mythread.status is not 'Completed'">
</cfloop>

<cfoutput>
The thread completed in
#mythread.elapsedTime#ms and generated the
following output: <br />
#mythread.output#
</cfoutput>
```

Joining Threads

- There may be a set of processes that you want to run asynchronously, yet have control ultimately return to the calling page once all have completed in order to output a combined result.
  - For example, you could join multiple <cffeed> requests so that they execute in parallel, returning their information interactively to the calling page.
  - In order to accomplish this, you will use the <cfthread action="join"> syntax as indicated below:

```
<cfthread
   action="join"
   name="thread1,thread2,….thread n"
   [optional]
   timeout="timeout in ms">
```

- In the following example, three <cffeed> operations are executed concurrently to parse files named feed1.xml, feed2.xml, and feed3.xml:

```
<cfset baseurl =
"http://localhost:8500/acfd9/assets/feeds/">

<cfset atitles=["Wash Post","WUSA","FOX News"]>

<cfloop from="1" to="3" index="i">
  <cfthread action="run"
            counter="#i#"
            name="thread#i#">

   <cffeed action="read"
           source="#baseurl#feed#counter#.xml"
           query="thread.qfeed" />

  </cfthread>
</cfloop>

<cfthread
 action="join"
 name="thread1,thread2,thread3"
 timeout="5000" />

<cfform>
 <cflayout type="tab" tabheight="200">
   <cfloop from="1" to="3" index="i">
    <cflayoutarea title="#atitles[i]#">
     <cfset thequery =
            cfthread["thread#i#"].qfeed>
     <ul>
     <cfoutput query="thequery">
     <li>
```

```
      <a href="#thequery.rsslink#"
target="_blank">#thequery.title#</a><br />

      #thequery.content#
      </li>
      </cfoutput>
      </ul>
      </cflayoutarea>
    </cfloop>
  </cflayout>
</cfform>
```

Temporarily stopping thread execution

- You can temporarily pause thread execution by using the `<cfthread action="sleep">` tag.
  - Typically this would be deployed in instances where a primary thread has a dependency whereby a secondary thread must complete, before the primary can finish executing.
- `<cfthread action="sleep">` performs essentially the same function as the `ColdFusion sleep()` function.
- General syntax for putting a thread into a wait-state:

```
<cfthread action="sleep" duration="milliseconds">
```

Note: While the thread is asleep, waiting threads will NOT be dequeued. The sleeping thread will continue to count against the limit of number of simultaneously executing threads, as stipulated in the ColdFusion Administrator.

Terminating Threads

- Typically you would use `<cfthread action="Terminate">` to end the processing of hung threads or in situations where you have detected a deadlock condition.
- In order to accomplish this, you would likely create a monitoring thread that would check the status of a targeted thread every n milliseconds, ultimately making a decision about whether termination is necessary.
- When spawning multiple threads, note that they should not execute shared resources simultaneously such an FTP connection.

Note: Be sure touse `<cflock>` where appropriate in order to isolate race conditions.

- In the following example, the thread dependentThread is monitored by the thread monitor and the dependent thread is terminated after 10 seconds of execution.

```
<cfthread name="dependentThread" action="run">
```

```
 <cfloop condition="true">
   <!--- infinite loop --->
 </cfloop>

</cfthread>

<cfthread name="monitor" action="run">
 <cfset time = 0>
 <cfloop condition="true">
  <cfif dependentthread.status is "RUNNING" and
                                    time ge 10000>

   <cfthread action="terminate"
             name="dependentThread" />

   <cfbreak>
  <cfelse>
   <cfset time += 500>
   <cfset sleep(500)>
  </cfif>
 </cfloop>
</cfthread>
```

Limiting the number of concurrently executing threads

- You can limit the number of concurrent threads executed by the server through the ColdFusion Administrator.
- Available on the Request Tuning page in the Tag Limit Settings section.
- The goal of limiting threads is to not have so many concurrent executing requests that the server's CPU gets time-sliced to the point of completing processes in an untimely fashion.

Monitoring and Aborting Threads

- You can monitor your threads and terminate them, if necessary, through the ColdFusion Server Monitor.
  - Access this feature through the ColdFusion Administrator -> Server Monitoring -> Server Monitor -> Launch Server Monitor -> Statistics->Active ColdFusion Threads.

Walkthrough 3: Creating Asynchronous Processes

In this walkthrough, you will perform the following tasks:

- Decrease the amount of time required to read, parse, and output three RSS feeds.

Steps

1. Using ColdFusion Builder, open /unit8/walk3/walk8-3.cfm and review the code with your instructor.

   Note: If your machine is connected to the internet, uncomment the <cffeed> tags from lines 36-55 and comment out the <cffeed> tags on lines 19-33

2. Where indicated by the comments, insert a <cftimer> block with the following attributes:
   - type : debug
   - label : Process Feeds
3. Browse the file and note the execution time indicated by the <cftimer> block.
4. Wrap the first `<cffeed>` tag on the page with a `<cfthread>` block. Use the following attributes:
   - action: run
   - name: feed1
5. Wrap the second `<cffeed>` tag on the page with a `<cfthread>` block. Use the following attributes:
   - action: run
   - name: feed2
6. Wrap the third `<cffeed>` tag on the page with a `<cfthread>` block. Use the following attributes:
   - action: run
   - name: feed3
7. Directly underneath the `<cfthread>` block that you inserted from the prior step, insert another `<cfthread>` tag that will join the three feed threads to the current page. Use the following attributes:
   - action: join
   - timeout: 8000
   - name: ”feed1,feed2,feed3”
8. Save the file and browse. Why are you receiving an error message?
9. Return to ColdFusion Builder

10. In each of the <cffeed> tags, prefix the name of the query variable with "thread." and remove the numeric suffix. Use the following example as a guide:

```
<cffeed action="read"
  source="#baseurl#feed1.xml"
  query="thread.qfeed">
```

11. Where indicated by the comment, modify the <cfset> tag to reference the variables created by the threads:

```
<cfset thequery = cfthread["feed#i#"].qfeed>
```

12. Save the file and test. Note the change in execution time. Depending on the available bandwidth, you might get an error reporting that one of the query variables does not exist. Why would this occur?

Caching Recordsets

- Database interaction is one of the most expensive tasks of most web applications.
- Reducing the number of queries and their frequency of execution will typically provide more benefit than any other single optimization.
- Recordsets can be cached in three ways:
  - `<cfquery>` attributes cachedwithin and cachedafter
  - Storing query results in a Session, Application, or Server variable
  - Using <cfcache> (covered later in this unit)

Query-based Caching

- Use query-based caching when you want a particular query cached for a specific period of time.
- To implement this type of caching, use one of the following `<cfquery>` attributes:
  - cachedwithin To specify a relative cache time
  - cachedafter To specify an absolute cache time
- Benefits of query-based caching:
  - Can dramatically enhance performance
  - Simple implementation
  - Well-suited for dynamic queries (query-by-example interfaces)
- The principal drawback, however, is that depending on your specific use-case scenario, you may use an unknown and potentially fatal amount of memory.
- Adobe recommends using query-based caching only when your query execution time exceeds 16ms.

cachedwithin attribute

- Use the cachedwithin attribute to specify the time period for which the query should be cached.
- The following query uses the same data (Homicides reported in the last week) for one full day before it is refreshed from the database:

```
<cfquery name="qGetHomicides"
   cachedwithin="#CreateTimeSpan(1,0,0,0)#">
```

```
    SELECT *
    FROM Crime
    WHERE offense='Homicide'
    and reportdatetime >=
      <cfqueryparam
                cfsqltype="cf_sql_date"
                value="#dateadd("ww",-1,now())#">
    ORDER BY reportdatetime
</cfquery>
```

Note: The arguments of the CreateTimeSpan( ) function are (days, hours, minutes, seconds).

cachedafter attribute

- Use the `cachedafter` attribute when the query should be refreshed at a certain time each day.
- If large volumes of database writes are regularly performed between midnight and 2:00 AM, you could refresh the query at 2:00 AM every day, and use that cached query until 12:00 AM the next day.
- These functions are often used with the cachedafter attribute: `CreateDateTime( )`, `Year( )`, `Month( )`, and `Day( )`.
- Here is the code for a `<cfquery>` tag to cache the query and update it at 2:00 AM every day:

```
<cfquery
     name="qGetHomicides"
     cachedafter ="#CreateDateTime(Year(Now()),
                                   Month(Now()),
                                   Day(Now()),
                                          2,0,0)#">
```

Note: The cachedafter attribute in this code breaks the day into two parts. From midnight to 2:00 am, no caching is done. The first time the query is used after 2:00 am, the query is cached and that query is used until midnight that day.

Flushing cached queries

- You can either flush the cache generated for a specific SQL statement, or you can simultaneously flush the cache for all cached queries in your application

Flushing an individual query

- You can immediately refresh the contents of a query that uses the cachedwithin or cachedafter attribute using all zero values in the `CreateTimeSpan()` function, as shown here:

```
<cfquery
      name="qGetHomicides"
      cachedwithin="#CreateTimeSpan(0,0,0,0)#">

  SELECT *
   FROM Crime
   WHERE offense='Homicide'
   and reportdatetime >=
     <cfqueryparam
               cfsqltype="cf_sql_date"
               value="#dateadd("ww",-1,now())#">
   ORDER BY reportdatetime
</cfquery>
```

Note: The query name, datasource, and resultant query string must be exactly the same (including any use of white space) as the code that originally cached the query.

Flushing all cached queries

- You can simultaneously flush all cached queries using the `<cfobjectcache>` tag. In the following example, all query caches are flushed across the application:

```
<cfobjectcache action="clear">
```

Query Caching Behavior

- When a query is cached, it can be referenced by any query that uses all of the same query information:
  - The values of the name and datasource attributes must be the same, and are case sensitive.
  - Resulting SQL statements must be the same, including spaces, returns, tabs, and so on. In addition, the case of all SQL statements must also be the same. For this reason, you should try to centralize your `<cfquery>` calls within reusable CFC methods.
  - If any of this information differs, ColdFusion does not use the cached query.
  - You cannot use `<cfqueryparam>` in a cached query prior to ColdFusion 8.

Caching queries in a persistent scope

- You can also store query data persistently as a session, application, or server-scoped memory variable.
- In order to accomplish this, simply prefix the name of your query with the name of one of ColdFusion's persistent scopes, as indicated below:

```
<cfquery name="application.offenseTypes">
 select distinct offense
 from crime
 order by offense
</cfquery>
```

- In order to refresh the contents of the recordset, you re-execute the query, using the same query name as before. There is no specific timeout, other than the timeouts for all application and session scoped variables that you set within your Application.cfc file or within the ColdFusion Administrator.
- Another strategy that you could employ is to instantiate the query into a variable that is globally scoped to a CFC. If the CFC is instatiated into an Application-scoped variable then the query will also be cached implicity.
- You can stop your application, thereby clearing all application variables, by using the ColdFusion 9 function `applicationStop()`. The application is restarted on the next request to the application.

Walkthrough 4: Caching Query Results

In this walkthrough, you will perform the following tasks:

- Apply query-based caching by using <cfquery>'s cachedwithin attribute.
- Cache a query implicitly within a cached ColdFusion component.

Steps

Cache a query with CachedWithin

1. In ColdFusion Builder, open /walk/unit8/walk4/components/crime.cfc.
2. Locate the `getCrimeTypes()` method.
3. Where specified by the comment, modify the query attributes to assign a name of "qOffenseList" to the query and cache it for one hour. Your code should appear similar to the following:

```
local.queryService.setcachedwithin(
                    createtimespan(0,1,0,0)
);

local.queryService.setName("qOffenseList");
```

Note: Cached queries in CFSCRIPT must be named

4. Save the file
5. In ColdFusion Builder, open /walk/unit8/walk3/walk8-4.cfm and review its contents.
6. Browse the file. Note that the first execution of the query is uncached. The second is cached.
7. Refresh the page in the browser. You should see that both instances of the query were cached.

Cache a query as a Component Property

8. In ColdFusion Builder, return to crime.cfc
9. Comment out the the code that you inserted from step 3
10. On line 20, add an additional property definition for a new property named qCrimeTypes

```
property type="query" name="qCrimeTypes";
```

11. Inside the init() method, before the `return` statement, invoke the setQCrimeTypes method, passing in the result from the getCrimeTypes() method as an argument:

```
setQCrimeTypes(getCrimeTypes());
```

12. Save the file
13. In ColdFusion Builder, open walk8-3a.cfm and review its contents with your instructor
14. Browse the file. Note that while the second <cfdump> does not indicate that the query was cached, the actual sql request only ran once on this page. Verify that the query ran once by reviewing your debug output.

Cache the component

15. In ColdFusion Builder, open /walk/unit8/walk4/Application.cfc
16. In the `onApplicationStart()` method, just above the `<cfreturn>` statement, cache the Crime.cfc component in a variable named application.cfc.specials as indicated below:

```
<cfset application.cfc.Crime =
createObject("component",
"#application.settings.cfcPath#Crime").init(
  application.googlemapkey,
  0,
  "1400 16th Street NW",
  "Washington",
  "DC",
  0.1)>
```

17. Save the file
18. Return to walk8-4.cfm
19. Delete the line of code that instantiates the Crime component
20. Change the `<cfdump>` tags to the following:

```
<cfdump
var="#application.cfc.crime.getQCrimeTypes()>
```

21. Save the file and browse with a url variable of `?init`
22. Re-browse the file without the url variable. Note that the code still executes, but no query activity is reported.

Caching Content with <cfcache>

- In every web application there are a number of pages whose contents do not change on each page request.
- In these instances, you might consider the following strategies:
  - Cache the entire page, or a large section of the page to disk with the <cfcache> tag.
  - Cache a specific section of a page to memory

Introducing the newly enhanced <cfcache>

- The `<cfcache>` tag enables you to store generated content on either the server's hard disk or server memory (the default)
- `<cfcache>` is now based on Ehcache - a widely used java distributed cache for general purpose caching (ehcache.org)
- The tag can create new cache files based on the requested url, including URL variables (e.g. index.cfm?id=5 is cached separately from index.cfm?id=6)
- `<cfcache>` was enhanced significantly in ColdFusion 9 to support the following:
  - Caching page fragments
  - Caching specific objects, including the ability to put, get, and flush cached objects
  - Setting cache dependencies
  - Setting an idle timeout
  - Getting metadata about cached objects
  - The ability to strip white space from cached page fragments
  - The ability to throw an exception if an error occurs when flushing a cached object

Working with <cfcache> attributes

Attribute	Actions	Default	Description
action	All	serverCache	• cache Server-side and client-side page caching • clientcache Browser-side page caching only. To cache a personalized page, use this option • flush Remove the current versions of cached page(s), fragment(s) or an object from the cache. The cache is refreshed the next time a user accesses the item. • get Retrieve an item from the cache • optimal Same as cache • put Add an object to the cache • serverCache Server-side caching only
dependson	cacheserveroptimal		A comma separated list of variables. If any of the variable values change, ColdFusion updates the cache. This attribute can take an expression.
directory	cache,servercache, clientCache,optimal flush,put	Cache in memory	Absolute path of cache directory
expireURL	Flush	Flush all cached pages	A URL reference. ColdFusion matches it against the mappings in the specified cache directory. Can include wildcards. For example: "*/view.cfm?id=*"
id	flush,get, put		The identifier for a cached object
idleTime	cache,servercache,clientCache, optimal, flush,put	No idle timeout	Flushes the cached object if it has not been accessed for the specified timespan: • A decimal number of days • A return value from createTimespan()

Attribute	Actions	Default	Description
metadata	get		The name of a structure variable in which to put object metadata. The get operation returns the following data: • timespan The time span during which the cached item is valid; the value of the timespan attribute for the cached item. • createdtime The time when the cache was created • lasthit The time when the cached item was most recently used. • hitnumber Number of times the cached item has been used. • missnumber Number of misses
name	get		The name of the variable in which to put the retrieved object
Password	cache,servercache clientCache,optimal flush		You must supply a password if the page to be cached sits behind basic authentication
Port	cache,servercache clientCache,optimal flush	The current page port	Specify if the http service is running on a unique port number
Protocol	cache,servercache clientCache,optimal flush	The current page protocol	Specify either http:// or https://
stripwhitespace	cache, servercache, optimal	false	Specifies whether to strip any unnecessary white space characters from a cached page fragment. Does not have any effect on cached pages or objects. the retrieved object.

Attribute	Actions	Default	Description
throwOnError	flush with ID attribute	false	A Boolean value specifying whether to throw an error if the flush action encounters an error. Otherwise the action does not generate an error if it fails. If this attribute is true you can handle the error in a cfcatch block, for example, if a specified id value is invalid.
Timespan		Cache expires only when <cfcache action=”flush”> is encountered	The interval until the page is flushed from the cache. A decimal number of days. For example: -”.25”, for one-fourth day (6 hours) -”1”, for one day -”1.5”, for one and one half days A return value from the CreateTimeSpan function; for example “#CreateTimeSpan(0,6,0,0)#”
useCache	cache, servercache, optimal	true	Specifies whether to use caching for a page. This attribute can be useful during development., or you could use a function to predict when to use a cache based on the application state.
usequerystring	cache, servercache, optimal	false	If true, generates a template cache ID that includes the query string. This means, that a new template cahce is created whenever the query string changes. If set to true, the attribute dependson considers the URL parameters defined in the querystring as well to generate template caches.
username	cache, servercache, clientcache, optimal, flush		Provide this if the page being cached or flushed requires authentication at the web server level.
value	put		The object to cache

- The following example retrieves data from the ColdFusion cache

```
<cfcache action="get" name="aFeeds"
                      id="aFeedsCache">

<cfif not isdefined("variables.afeeds")>

  <cffeed action="read"
          source="#baseurl#feed1.xml"
          query="qfeed1" />
```

```
    <cffeed action="read"
            source="#baseurl#feed2.xml"
            query="qfeed2" />

    <cffeed action="read"
            source="#baseurl#feed3.xml"
            query="qfeed3" />

    <cfset aFeeds = [qfeed1,qfeed2,qfeed3]>

    <cfcache action="put"
             value="#aFeeds#" id="aFeedsCache">

</cfif>
```

Custom Caching Mechanisms

- In many instances, you might find that the most efficient approach for caching your site involves using a combination of <cfcache>, query-based caching, application variable caching, and your own custom algorithms.
- You might, for example, want to generate static HTML files from your CFM's to achieve the optimal scalability benefit possible. Static htm files are almost infinitely scalable since there is no dependency on application or database servers for server-side processing. Today's web servers can easily accommodate millions of page views per day, assuming that no server-side processing is required. This approach does, however, limit interactivity.
- Alternately, you may adopt a hybrid approach where the majority of the page is rendered statically, but small areas are left to execute on each request. As you devise the caching strategy for your ColdFusion application, consider the following:
  - Does the information presented on your site need to be updated on each request? Caching results for even short periods of time on high-traffic sites can yield big returns.
  - There is always a performance cost for cache misses. Due to their nature, <cffile>, <cfhttp>, <cfdirectory>, <cfdocument>, <cfpresentation> are all performance bottlenecks as compared to the execution speed of other ColdFusion tags.
  - Query caching and <cfcache> , if not used judiciously, can consume vast amounts of server memory. As we learned earlier in this unit, server memory is not an infinite resource. As you apply caching techniques, monitor the memory usage on your site through the techniques described earlier in this course.

- As you test your application, be sure to work with a representative data set. The costs (and relative benefits) of query-based caching are going to grow as the data volume grows in your database.
- Caching mechanisms exist outside the realm of ColdFusion. The third-party service Akamai (www.akamai.com) provides content caching and is used by virtually every massively trafficked site on the Internet.

Walkthrough 5: Using <cfcache>

In this walkthrough, you will perform the following tasks:

- Use <cfcache> to cache RSS Feeds
- Programatically expire the cache.

Steps

Use <cfcache> to cache RSS Feeds

1. In ColdFusion Builder, open /walk/unit8/walk5/walk8-5.cfm
2. Where indicated by the comment, insert a <cfcache> tag with the following attributes:
    - action : get
    - name : afeeds
    - id : aFeedsCache
3. Wrap the three <cffeed> tags with a <cfif> block that evaulates whether or not variables.afeeds is defined:

```
<cfif not isdefined("variables.afeeds")>

    <cffeed action="read"
            source="#baseurl#feed1.xml">

    ....

    <cfset aFeeds = [qfeed1,qfeed2,qfeed3]>

</cfif>
```

4. Directly above the </cfif> you inserted from the previous step, insert a <cfcache> tag with the following attributes:
    - action : put
    - value : #afeeds#
    - id : aFeedsCache
5. Save the file and browse. Note the page execution time in your debug output.
6. Refresh the page in your web browser. You should see a significant decrease in the amount of time required to process the feeds.
7. Return to ColdFusion Builder and review the contents of the page with your instructor. Discuss strategies for caching the entire page. When would it make sense to cache the entire page instead of just the RSS feeds?

Manage the Cache

8. Open walk8-5a.cfm

9. Where indicated by the comment, insert a <cfcache> tag with the following attributes:
    - action : get
    - metadata : stMetadata
    - id : aFeedsCache
    - name : aFeedsCache
10. At the top of the file, where indicated by the comment, insert a <cfcache> tag with the following attributes:
    - action : flush
    - id : aFeedsCache
11. Save the file and browse it. Note the details about the cache
12. Click the Expire Cache button
13. Browse walk8-5.cfm. Note that the file took longer to execute as the cache was refreshed.

Summary

- Scalability deals with how well your application supports multiple users simultaneously accessing your system.
- Performance is the overall responsiveness of the end-user experience.
- Robustness deals principally with how easily bugs can be fixed and new features deployed.
- ColdFusion is an application that runs on a J2EE application server.
- ColdFusion should not be used to host a production application without first tuning its various configuration settings.
- ColdFusion stability and performance is highly dependent on the performance characteristics of the Java Virtual Machine and database drivers.
- You can use a variety of tools to identify bottlenecks including, but not limited to, ColdFusion's debug output, the ColdFusion Server Monitor, and third-party load testing tools and services.
- You can cache queries in server memory using cachedwithin/ cachedafter.
- You can cache generated content and data using `<cfcache>`
- You can significantly affect performance by creating asynchronous processes with `<cfthread>`

Review

1. Define the differences between performance, scalability, and robustness.

2. What is the methodology for determining the optimal number of simultaneous requests that ColdFusion should handle?

3. How does reviewing the debug output from a page help you identify potential bottlenecks?

4. Describe two methods for caching query data.

5. How can you flush a cached query?

6. How can you cache the generated output from an entire page request?

7. What ColdFusion tag is used to instantiate an asynchronous process?

8. Describe the purpose of `<cfthread action=”join”>`