msg talk:
ColdFusion Performance Tuning

.msg at.NET

.consulting .solutions .partnership

Marcus Raphelt
04.10.2004

Introduction

.msg at.NET

1. **Introduction**
2. **Server Settings**
3. **Code Optimization**
   (real world examples)

Introduction

Marcus Raphelt – whoami

- CF Programmer since 1998 (started with CF 3.1)
- Author for several CFML-Websites
- Macromedia Certified Professional (ACCF5D, ACCFMXD, CCFI)
- Contact: raphelt@msg-at.net
- "Agent M" on bloginblack.de :-)

Server Settings

Setting optimum values in CFADMIN

- Limit simultaneous requests to
  - 3 per CPU, but depends on your application:
  - ➢3 if your application is rather CPU-intensive
  - ➢4-6 if you mainly call external processes
- Timeout requests after…
  - Highly recommended, but no general rule for the value – I'd suggest 60
- Restart at x unresponsive requests
  - Here, I suggest the same value as above
- Suppress whitespace by default
  - Should be activated: reduces web server traffic
- Enforce Strict Attribute Validation
  - Should be activated

Server Settings (continued)

Setting optimum values in CFADMIN

- Template Cache size

  Should be set to the total size of all you CF Templates

- Trusted Cache

  Depends!

- Limit cached database connection inactive time

  Depends!

- Maximum number of cached queries

  Depends!

- Store Client Variable Data in a database
- Deactivate „Single Threaded Sessions“
- Variable Scope Lock Settings: set all to „Full Checking“
- Optional: use IP addresses instead of FQNs

Code Optimization

Variable Locking

- If you are still using CF 5 or older, ensure that every access to shared scope variables is locked. If not, the server may become unstable under heavy load
- In CFMX, this became obsolete

Optimize your SQL queries

- select * is evil!
- Only select the columns you really need
- Cache your queries as long as possible

Try to avoid using evaluate() whenever possible

- Under some circumstances, you can avoid using evaluate():

  evaluate("form.textfield_#counter#") is the same as

  form['textfield_#counter#']

Hash (#) overuse

•Very often, I've seen code snippets like these:

```
<cfset blah = "#listGetAt(#form.list#,'#counter#')#">

<cfif #form.theField# eq #currentRow#>
```

Of course – this code works, but in functions, tags etc., CF evaluates automatically. Besides being more readable, the following code is even faster:

```
<cfset blah = listGetAt(form.list, counter)>
<cfif form.theField eq currentRow>
```


CFHTTP without timeout

CFHTTP is an external process. Once started, CF waits for it to return.
If no timeout value is given…

•…CF kills the thread after the "Timeout requests after … " value is reached. If you set 60 seconds here, one of CF's requests pipes will be blocked for a minute! Imagine what just **ONE** impatient reload-reload-reload-reload-visitor could do to your server…

• If you do not enable the timeout option in CFADMIN, CF keeps this external request up until the next service restart (!).

```
<cfhttp url="http://www.bloginblack.de"
        timeout="8"
>
```

Always set that!

Setting variables

You can even gain some execution time by setting variables the right way:

• As mentioned before, don't overuse hashes:

```
<!--- not a single hash in here… --->
<cfset theLen = len(trim(left(inputString,3)))>
```

• 1 – 2 variables: use a normal CFSET

```
<cfset agentK = "kai">
<cfset agentM = "marcus">
```

• 3 or more variables: use CFSCRIPT

```
<cfscript> <!--- that's about 20% faster --->
        agentK = "kai";
        agentM = "marcus";
        website = "bloginblack.de";
</cfscript>
```


Scoping variables

Although it's more code to type :-), always scope your variables!
This has several reasons:

• The code will be easier to read (and maybe understand)

If you call a variable like this `#lastName#`, CF has to loop through all variable scopes (Local/Query => CGI => File => URL => Form => Cookie => Client) until it finds the appropriate item. Have a look at this example:

Let's say `saveValues.cfm` receives a form post. You save the data like this:

```
<cfquery name="saveValues" datasource="theKingFanClub">
  insert into members(firstName, lastName)
  values('#firstName#', '#lastName#')
</cfquery>
```

As no scope is given, CF finds this value in the fifth of seven possible scopes.
If you tell CF in which scope your desired variable resides, you gain execution time.

Loops

Array Loops
You can optimize array loops by assigning the arrayLen to a variable. Normally, you would code an index loop like this:

```
<cfloop from="1" to="#arrayLen(theArray)#" index="x">
… <!--- wouldn't you? :-) --->
</cfloop>
```

The disadvantage of this technique is that CF evaluates the function arrayLen() for each array position. This one is better:

```
<cfset theArrayLen = arraylen(theArray)>
<cfloop from="1" to="#theArrayLen#" index="x">
… <!--- way to go --->
</cfloop>
```

List loops
You should consider converting a list to an array before looping over it. In some cases, this results in better performance (especially for long lists).

Loops (continued)

CFLOOP or CFOUTPUT?
I did a lot of measuring to find out which way of looping is faster. As far as loops are concerned, there are some differences between CF 4 / 5 and CFMX:

- Prior to CFMX, CFLOOP was generally faster.
- Since CFMX, I couldn't figure out a noticeable difference.

Despite that both perform equally, I'd still recommend using them this way:

- Use a CFOUTPUT loop if you really OUTPUT something to the browser
- Use a CFLOOP loop for "pure" calculation

Boolean evaluation

One of CF biggest advantages is the ability to regard nearly everything as a boolean value. As boolean evaluations are fast, try to use them as often as possible. To CF, everything besides "0" (zero) is TRUE. The "0" itself is FALSE.

```
<cfif queryName.recordcount> <!--- no need for gt 0! --->
</cfif> <!--- if recordcount is >0, it's TRUE!--->

<cfif isDefined("form.blah")> <!--- no EQ "true" here!--->
</cfif> <!--- (i've seen that very often…) --->

<cfif len(theString)>
</cfif> <!--- 2x faster than ' theString NEQ "" ' --->

<cfif not compareNoCase(string1, string2)>
</cfif> <!--- 2x faster than "string1 EQ string2"! --->

<cfif findNoCase("blah", theString)>
</cfif> <!--- 2x faster than ' thestring contains "blah" '--->
```

As a general rule, functions() tend to be faster than operators (EQ, NEQ, CONTAINS etc.).

Boolean evaluation (continued)

An overview of all possible forms of boolean values:

TRUE / FALSE

YES / NO

TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
…	-2	-1	0	1	2	…

Query of queries

If you have to query one table several times during a page request, try using QoQ to minimize the number of database requests:

```
<cfquery name="theFanBase" datasource="theKingFanClub">
   select firstName, lastName, ID, […] registerDate
   from thatTable
</cfquery>

<cfloop from="1" to="1000" index="x">
   <cfquery name="timeEater" dbtype="query">
        select firstName, lastName
        from theFanBase
        where id = #x#
   </cfquery>
</cfloop>
```

In the above example, there are 1001 queries, but only one actually requests data from the database.

Query of queries (continued)

Here's another cool QoQ example I don't want to withhold:

```
<cfif not isdefined("request.binaryContents")>
       <cfdirectory name="request.binaryContents" action="list"
       directory="c:\apache\htdocs\frontend\images">
</cfif>

<cfquery name="getFileSize" dbtype="query">
   select size from request.binaryContents
   where name = '#form.name#'
</cfquery>
```

Even if you need your image folder's content 1000 times during the page request (meaning ALL templates, including custom tags and includes!), CF will actually read from the harddisk just once!

cf_supercache

One of my favourite (if not the favourite) techniques to speed up CF applications is RAMCaching, or SuperCaching. It is suitable for almost any kind of template and really really really BOOSTS up you application. It's no exaggeration to say that you in fact can bring down a timeEater's execution time from 50000 ms to 10 ms.

How can this be achieved?

Well - cf_supercache cannot conjure - it keeps the generated content in the server's RAM for a specified timespan. So, the first request is still slow, but subsequent page requests will get their data directly out of the RAM, bypassing all processing logic.

cf_supercache

Let's say we have the following code:

```
<!---do some very cpu-eating stuff--->
<cfquery name="aVeryBigQuery" ...>
</cfquery>

<cf_doSomeTimeConsumingThings>

<cfoutput query="aVeryBigQuery">
  <a href="index.cfm?dsn=theKingFanClub&firstname=#fn#">..</a>
</cfoutput>
```

cf_supercache

Just wrap your code in cf_supercache tags - you're done!

```
<cf_supercache
  name="unique_cache_id_for_this_page_or_content_snippet"
  expires="#createtimespan(0,1,0,0)#"
  stripWhitespace="true">

<!---do some very cpu-eating stuff--->
<cfquery name="aVeryBigQuery" ...>
</cfquery>

<cf_doSomeTimeConsumingThings>

<cfoutput query="aVeryBigQuery">
  <a href="index.cfm?dsn=theKingFanClub&firstname=#fn#">..</a>
</cfoutput>

</cf_supercache>
```


cf_supercache

Just wrap your code in cf_supercache tags - you're done!

```
<cf_supercache
  name="unique_cache_id_for_this_page_or_content_snippet"
  expires="#createtimespan(0,1,0,0)#"
  stripWhitespace="true">

<!---do some very cpu-eating stuff--->
<cfquery name="aVeryBigQuery" ...>
</cfquery>

<cf_doSomeTimeConsumingThings>

<cfoutput query="aVeryBigQuery">
  <a href="index.cfm?dsn=theKingFanClub&firstname=Zak">..</a>
</cfoutput>

</cf_supercache>
```

Questions?

.consulting .solutions .partnership