Fun with Hash Performance and Google Charts

Jason Rahm, 2011-04-01

Currently, I’m working on an update to Deb’s excellent tech tip on hash load balancing. During the prep stages, I started thinking about hashing algorithms in general and how resource intensive they are when used in iRules. Also, I’ve been a little jealous of Colin, Joe, & George and their creative use cases for the Google Charts API, so for the first entry of the New Year I thought I’d indulge myself with a little geekery.

The Algorithms

Several hashing algorithms are available for use on the LTM. First link is some background on each of the algorithms (or family of algorithms as is the case with SHA2) and the second link is the DevCentral wiki page for the algorithm’s use in iRules.

- CRC (crc32)
- MD5 (md5)
- SHA1 (sha1)
- SHA2
    - sha256
    - sha384
    - sha512

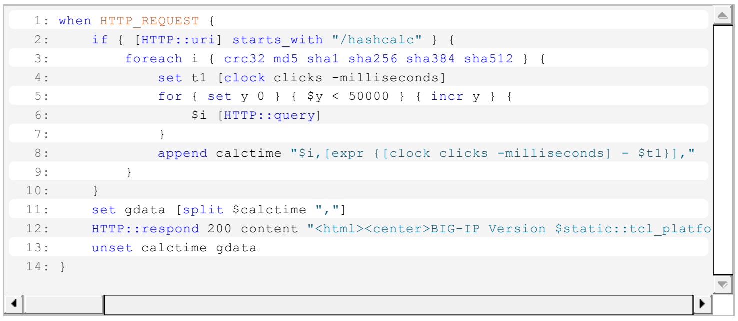
It might be of worth to note that the crc32 algorithm differs from the rest in that it is checksum function, whereas the rest of them are cryptographic functions. Checksum functions are primarily used for error detection and cryptographic functions primarily in security applications, but both of them can be used in the ordinary tasks of load balancing as well. There are pros/cons to resource utilization and distribution characteristics. I’ll just take a look at resources in this tech tip, but I’ll revisit distribution in the hash load balancing update I mentioned earlier. To give you an idea of the various digest/block sizes and the resulting output, see the table below. Note that the message in all cases is "DevCentral 2011."

Algorithm	Digest Size	Block Size	Message Digest
crc32	9	--	363676975
md5	16	64	ba4b3fbd19b62bff3fbd5fb78e44e267
sha1	20	64	5a1cd3e3cd4a14aea459d507ee233ba2dd893793
sha256	32	64	4296905615d6cbd47910aabda41016c029d1a0a7a438eb32def4c723c11554ca
sha384	48	128	3487da09d585e164a2a8455252dfa519660f0886481dd9534bbbb6fc2c94897d7436689d30afb0d48a22b159f8498003
sha512	64	128	08dbfdd505871f1d05c16bee2a1696419c22ebd9123793ca1fffed79da9d567df8827173614db562d006d9b3c19ce189a09cf38cb9f0d38310085efc849c274e

Note: Data above actually generated from python zlib and hashlib libraries on Ubuntu 9.04. Just a representative look at the differences in hashing algorithms.

The iRule

The code is below. Note that the iRule expects a path of /hashcalc and a query (which it uses as the source of the hash computation). If you wanted to pass the number of computations to the iRule in the query, that would be a very small modification.

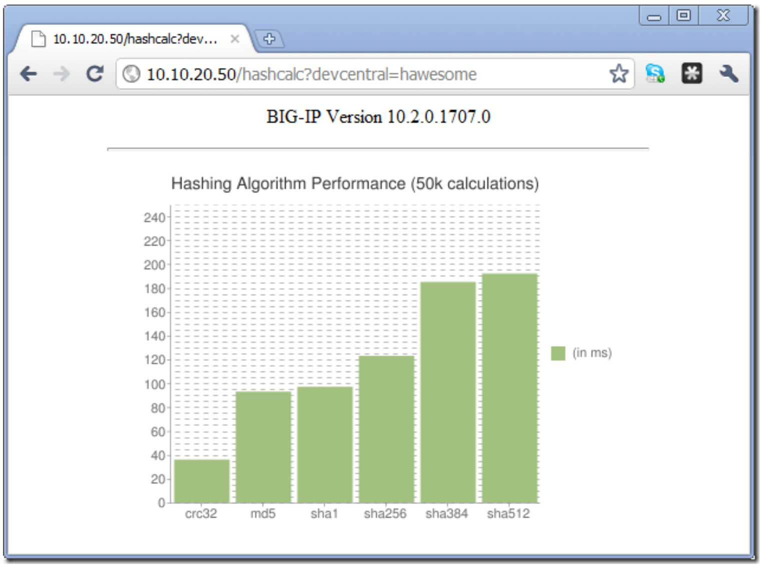


```
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/hashcalc" } {
        foreach i { crc32 md5 sha1 sha256 sha384 sha512 } {
            set t1 [clock clicks -milliseconds]
            for { set y 0 } { $y < 50000 } { incr y } {
                $i [HTTP::query]
            }
            append calctime "$i,[expr {[clock clicks -milliseconds] - $t1}],"
        }
    }
    set gdata [split $calctime ","]
    HTTP::respond 200 content "<html><center>BIG-IP Version $static::tcl_platfo
    unset calctime gdata
}
```

I made sure each hashing algorithm ran enough times to plot out some meaningful numbers, settling in on 50k calculations, passing the iRules command in through the forearch loop and appending the calctime variable with the algorithm and milliseconds required to run the calculations.

The Results

The numbers, courtesy of HTTP::respond and a Google Charts bar graph:



10.10.20.50/hashcalc?dev...
10.10.20.50/hashcalc?devcentral=hawesome
BIG-IP Version 10.2.0.1707.0
Hashing Algorithm Performance (50k calculations)
240
220
200
180
160
140
120
100
80
60
40
20
0
crc32
md5
sha1
sha256
sha384
sha512
🟩 (in ms)

You can see that md5 takes more than twice the time as crc32 to compute the hash, that md5/sha1 are relatively even before stepping to sha256 and then finally to sha384/sha512, which are then roughly twice md5/sha1.

Conclusion

It was a fun investment to look at how the numbers played out between the hashing algorithms. Note that I ran this on a 3600 platform, your mileage may vary on different hardware (or in VE). If you run this, post your numbers back, I'd be curious to see the variance in platform and TMOS version.

Related Articles

- Hash Load Balancing and Persistence on BIG-IP LTM > DevCentral ...
- Election Hash Load Balancing
- Cookie-Hash not working - DevCentral - F5 DevCentral > Community ...
- Election Hash rule - DevCentral - F5 DevCentral > Community ...
- DevCentral Weekly Roundup | Audio Podcast - Hash Load Balancing
- Persistence cookie hash failed - DevCentral - F5 DevCentral ...
- DevCentral Weekly Roundup | Audio Podcast - Election Hash
- Algoritm for hash persistence - DevCentral - F5 DevCentral ...

Technorati Tags: F5 DevCentral,BIG-IP,LTM,iRules,hash algorithm,md5,crc32,sha1,sha2,Jason Rahm

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com