Go based content filtering software on FreeBSD

Ganbold Tsagaankhuu, Mongolian Unix User Group
Esbold Unurkhaan, Mongolian University of Science and Technology
Erdenebat Gantumur, Mongolian Unix User Group

AsiaBSDCon
Tokyo, 2015
Content

- Introduction
- Rationale behind our choices
- Related projects
- Experienced challenges
- Benchmark Case 1, 2 and results
- Conclusions and future works
Introduction

• What is the meaning of Shuultuur?

Шүүлтүүр

Подпись

Шүүлтүүр
Rationale behind our choices

• Why content filter?
  • Some control over unwanted content from web
    • Enforce security policies in corporates
    • Parental control
    • Schools
    • Libraries
  • Inappropriate content depending from age
    • Adult
    • Violence
    • Drugs etc.
Rationale behind our choices

• Why Go?
  • Fast, lightweight, easy to prototype
  • Productive
  • Performance
Rationale behind our choices

- **Why Go?**
  - Go is
    - Compiled, statically typed
    - Garbage collected
    - Object oriented
- **Performance of Go’s**
  - Somewhat comparable to C
  - Better than some of interpreted languages
- **Concurrency**
  - Part of the programming language features
  - It has strong support for multiprocessing
Rationale behind our choices

• Why Go?
  • Go includes multiple useful built-in data structures such as maps and slices
  • Goroutines and channels
    • A goroutine is a function executing concurrently with other goroutines in the same address space.
    • It is lightweight and communicates with other goroutines via channels
    • In contrast coroutines communicate via yield and resume operations
  • Built-in profiling tool
  • Extensive number of libraries
  • BSD licensed
Rationale behind our choices

• Why FreeBSD is platform of choice?
  • Powerful, mature and stable
  • Complete, reliable and self-consistent distribution
  • FreeBSD’s networking stack is very solid and fast
  • Easy to install and deploy the necessary applications and software using port and package system
  • Making custom FreeBSD image easily (such as NanoBSD)
  • We love FreeBSD
Related projects

- **goproxy**
  - Customizable HTTP proxy library for Go.
  - Supports regular HTTP proxy,
  - HTTPS through CONNECT,
  - "hijacking" HTTPS connection using "Man in the Middle" style attack
  The intent of the proxy is to be usable with reasonable amount of traffic yet, customizable and programmable

- **gcvis**
  - Visualizes Go program gctrace data in real time

- **profile**
  - Simple profiling support package for Go

- **go-nude**
  - Nudity detection with Go
Related projects

- **xxhash-go**
  - Go wrapper for C xxhash - an extremely fast Hash algorithm
  - Working at speeds close to RAM limits

- **powerwalk**
  - Go package for walking files
  - Concurrently calling user code to handle each file

- **redigo**
  - Go client for the Redis database

- **Redis**
  - Open source, BSD licensed, advanced *key-value cache* and *store*
Experienced challenges

• Problems during development:
  • The Shallalist blacklist
    • 1.8 million URL/Domain entries.

...  
// Store URL/Domains as a key and  
// category as a value  
conn.Do("SET", urls_or_domain, category)  
...
Experienced challenges

- Solution. Changed the code to:

```go
// use xxhash to get checksum from URL/Domain
blob := []byte(url_or_domain)
h32g := xxh.GoChecksum32(blob)

/*
 * Store it as hash in Redis in following way:
 *   key   = 0xXXXX (first half of URL/Domain),
 *   field = XXXX   (second half of URL/Domain),
 *   value = category
 */
hash_str := fmt.Sprintf("0x%08x", h32g)
key      := hash_str[0:6]
value    := hash_str[6:]
conn.Do("HSET", key, value, category)
...```
Experienced challenges

• Banned and weighted phrase lookup problem
  • Problem: Storing all phrases in Redis
    • Slow and not efficient
    • Loop is expensive
  • Solution: Graph and map
    • Every unique word is an edge of the graph
    • Edges and Vertices are stored in the map
      • Map – Go’s implementation of hash table
  • Problem: Regular expression based search
    • CPU intensive
  • Solution: Graph and Boyer Moore search algorithm
Experienced challenges

Graph representation

For example: “sex woman”, “sex man” and “drunk woman sex” words in Graph.

Man: 2-1
Sex: 2-1, 2-3, 4-3-2
Drunk: 4-3-2
Woman: 2-3, 4-3-2
Experienced challenges

- Reading HTTP response bodies into memory
  - Heap memory usage grow very large
    - Lots of allocations
    - When the rate of connections per second is high

- Solution
  - Streaming parser by utilizing the io.Reader interface
  - Limiting incoming requests
  - CPU and memory profiling
    - Go’s built-in profiler pprof
Experienced challenges

# go tool pprof --alloc_space ./shuultuur_mem /tmp/profile228392328/mem.pprof
Adjusting heap profiles for 1-in-4096 sampling rate
Welcome to pprof! For help, type 'help'.
(pprof) top15
Total: 11793.7 MB

3557.7  30.2%  30.2%  3557.7  30.2% runtime.convT2E
1212.1  10.3%  40.4%  1212.1  10.3% container/list.(*List).insertValue
832.3   7.1%  47.5%  2434.8  20.6% github.com/garyburd/redigo/redis.(*conn).readReply
807.9   6.9%  54.4%  1874.6  15.9% github.com/garyburd/redigo/redis.(*Pool).Get
673.8   5.7%  60.1%  673.8   5.7% github.com/garyburd/redigo/redis.Strings
544.5   4.6%  64.7%  549.4   4.7% main.regexBannedWordsGo
521.1   4.4%  69.1%  521.1   4.4% bufio.NewReaderSize
490.9   4.2%  73.3%  490.9   4.2% bufio.NewWriter
438.2   3.7%  77.0%  438.2   3.7% runtime.convT2I
369.8   3.1%  80.1%  7622.9  64.6% main.workerWeighted
255.0   2.2%  82.3%  255.9   2.2% main.regexWeightedWordsGo
235.5   2.0%  84.3%  235.5   2.0% bytes.makeSlice
229.9   1.9%  86.2%  397.1   3.4% io.Copy
168.3   1.4%  87.6%  168.3   1.4% github.com/garyburd/redigo/redis.String
162.6   1.4%  89.0%  4048.9  34.3% main.getHkeysLen

(pprof)
Experienced challenges

# go tool pprof --alloc_space .:/shuultuur /tmp/profile287823990/mem.pprof
Adjusting heap profiles for 1-in-4096 sampling rate
Welcome to pprof! For help, type 'help'.
(pprof) top30
Total: 2156.3 MB

596.9 27.7% 27.7% 1066.4 49.5% io.Copy
406.3 18.8% 46.5% 406.3 18.8% compress/flate.NewReader
113.5 5.3% 60.0% 115.4 5.4% code.google.com/p/go.net/html.
(*Tokenizer).Token
  78.3 3.6% 63.6% 78.3 3.6% code.google.com/p/go.net/html.
(*parser).addText
  68.4 3.2% 66.8% 68.4 3.2% strings.Map
...

37.7 1.7% 78.9% 736.6 34.2% main.ProcessResp
27.9 1.3% 80.2% 27.9 1.3% makemap_c
...
12.8 0.6% 91.8% 44.5 2.1% bitbucket.org/hooray-976/shuultuur/db.GraphBuild
12.5 0.6% 92.4% 12.5 0.6% strings.genSplit
10.7 0.5% 92.9% 595.5 27.6% main.getContentFromHtml
...
Experienced challenges

- CPU usage

lastpid: 1189; load averages: 7.30, 2.42, 0.93 up 0+00:30:51 14:57:41
61 processes: 1 running, 60 sleeping
CPU: 20.5% user, 0.0% nice, 42.0% system, 6.6% interrupt, 31.0% idle
Mem: 104M Active, 63M Inact, 225M Wired, 234M Buf, 7502M Free
Swap: 16G Total, 16G Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1131</td>
<td>tsgan</td>
<td>22</td>
<td>52</td>
<td>0</td>
<td>182M</td>
<td>46196K</td>
<td>uwait</td>
<td>4</td>
<td>9:29</td>
<td>685.50%</td>
<td>shuultuur</td>
</tr>
<tr>
<td>900</td>
<td>redis</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>69952K</td>
<td>42512K</td>
<td>uwait</td>
<td>6</td>
<td>1:11</td>
<td>88.48%</td>
<td>redis-server</td>
</tr>
<tr>
<td>1130</td>
<td>tsgan</td>
<td>6</td>
<td>20</td>
<td>0</td>
<td>37856K</td>
<td>9084K</td>
<td>piperd</td>
<td>1</td>
<td>0:01</td>
<td>0.00%</td>
<td>gcvis</td>
</tr>
<tr>
<td>918</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>72136K</td>
<td>5832K</td>
<td>select</td>
<td>5</td>
<td>0:00</td>
<td>0.00%</td>
<td>sshd</td>
</tr>
<tr>
<td>889</td>
<td>squid</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>70952K</td>
<td>16412K</td>
<td>kqread</td>
<td>5</td>
<td>0:00</td>
<td>0.00%</td>
<td>squid</td>
</tr>
<tr>
<td>1049</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>38388K</td>
<td>5168K</td>
<td>select</td>
<td>11</td>
<td>0:00</td>
<td>0.00%</td>
<td>ssh</td>
</tr>
<tr>
<td>998</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>72136K</td>
<td>5904K</td>
<td>select</td>
<td>9</td>
<td>0:00</td>
<td>0.00%</td>
<td>sshd</td>
</tr>
<tr>
<td>919</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>17564K</td>
<td>3528K</td>
<td>pause</td>
<td>2</td>
<td>0:00</td>
<td>0.00%</td>
<td>csh</td>
</tr>
<tr>
<td>868</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>22256K</td>
<td>3284K</td>
<td>select</td>
<td>11</td>
<td>0:00</td>
<td>0.00%</td>
<td>ntpd</td>
</tr>
</tbody>
</table>
Experienced challenges

- CPU usage after optimizations

```sh
lastpid: 1253; load averages: 0.15, 0.31, 0.32 up 0+00:55:22 11:55:42
45 processes: 1 running, 44 sleeping
CPU: 1.4% user, 0.0% nice, 0.0% system, 0.0% interrupt, 98.6% idle
Mem: 96M Active, 72M Inact, 279M Wired, 310M Buf, 7445M Free
Swap: 16G Total, 16G Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1183</td>
<td>root</td>
<td>17</td>
<td>20</td>
<td>0</td>
<td>142M</td>
<td>37348K</td>
<td>uwait</td>
<td>0</td>
<td>7:28</td>
<td>14.31%</td>
<td>shuultuur</td>
</tr>
<tr>
<td>896</td>
<td>redis</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>78144K</td>
<td>62896K</td>
<td>uwait</td>
<td>3</td>
<td>0:52</td>
<td>0.00%</td>
<td>redis-server</td>
</tr>
<tr>
<td>1182</td>
<td>root</td>
<td>6</td>
<td>20</td>
<td>0</td>
<td>45048K</td>
<td>16840K</td>
<td>uwait</td>
<td>9</td>
<td>0:16</td>
<td>0.00%</td>
<td>gcvis</td>
</tr>
<tr>
<td>993</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>72136K</td>
<td>6744K</td>
<td>select</td>
<td>9</td>
<td>0:06</td>
<td>0.00%</td>
<td>sshd</td>
</tr>
<tr>
<td>1187</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>9948K</td>
<td>1600K</td>
<td>kqread</td>
<td>10</td>
<td>0:03</td>
<td>0.00%</td>
<td>tail</td>
</tr>
<tr>
<td>1091</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>16596K</td>
<td>2548K</td>
<td>CPU8</td>
<td>8</td>
<td>0:02</td>
<td>0.00%</td>
<td>top</td>
</tr>
<tr>
<td>1204</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>38388K</td>
<td>5164K</td>
<td>select</td>
<td>5</td>
<td>0:00</td>
<td>0.00%</td>
<td>ssh</td>
</tr>
<tr>
<td>1196</td>
<td>tsgan</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>72136K</td>
<td>5904K</td>
<td>kqread</td>
<td>1</td>
<td>0:00</td>
<td>0.00%</td>
<td>sshd</td>
</tr>
<tr>
<td>885</td>
<td>squid</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>70952K</td>
<td>16384K</td>
<td>kqread</td>
<td>0</td>
<td>0:00</td>
<td>0.00%</td>
<td>squid</td>
</tr>
</tbody>
</table>
```

...
Experienced challenges

• Memory usage
Experienced challenges

- Memory usage after optimizations
Experienced challenges

- Other improvements
  - Learned mode (caching)
    - To not check HTTP response bodies every time
  - Rate limiting on incoming requests utilizing Redis
  - Limit the listener to accept a specified number of simultaneous connections
Experienced challenges

- Learned mode

  ...

  // Learn and store this URL to redisdb temporarily
  // use xxhash to get checksum from URL/Domain
  blob1 := []byte(requrl)
  h32g := xxh.GoChecksum32(blob1)

  // key = 0xXXXXXXXX for expire_time seconds,
  // 1 for BLOCK, 2 for PASS
  key := fmt.Sprintf("%s0x%08x", policy, h32g)

  // SET key value [EX seconds]
  // [PX milliseconds] [NX|XX]
  db.Exec("SET", key, BLOCK, "EX", EXPIRE, "NX")
  ...
  ...
Experienced challenges

- **Limit listener:**

```go
... type Server struct {
    *http.Server
    ListenLimit int // Limit the number of outstanding requests
}
func (srv *Server) ListenAndServe() error {
...
    l, err := net.Listen("tcp", addr)
    l = netutil.LimitListener(l, srv.ListenLimit)
    return srv.Serve(l)
}
...
if LISTEN_LIMIT_ENABLE == 1 {
    srv := &Server {
        ListenLimit: LISTEN_LIMIT,
    }
    log.Fatal(srv.ListenAndServe())
} else {
    log.Fatal(http.ListenAndServe(":8080", proxy))
}
```
Experienced challenges

- Slow image filtering on HTTP response
  - Used go-nude, but temporarily disabled until we find a proper solution
- High number of goroutines under heavy load
  - High CPU and memory usage.
  - Currently we are investigating the issue
Experienced challenges

• Problem: Our program panics sometimes with following message:
  • panic: dial tcp 127.0.0.1:6379: connection reset by peer

• Solution:
  • This was related to OS settings.
    • netstat -anL shows the limits.
    • Increased:
      • kern.ipc.somaxconn sysctl value
  • Increased tcp-backlog in redis.conf
Benchmark (Case 1)

- Test environment (Case 1):
  - Server OS
    - FreeBSD 9.2-RELEASE amd64
  - Server hardware:
    - CPU - Intel(R) Xeon(R) X5670 2.93GHz
    - Memory - 8192MB
    - FreeBSD/SMP -12 CPUs (package(s) x 6 core(s) x 2 SMT threads)
  - Go version 1.3.2
  - Dansguardian version 2.12.0.3
  - Squid version 3.4.8_2
Benchmark (Case 1)

- Increased some sysctl and /etc/sysctl.conf includes following:

```
kern.ipc.somaxconn = 27737
kern.maxfiles = 123280
kern.maxfilesperproc = 110950
kern.ipc.maxsockets = 85600
kern.ipc.nmbclusters = 262144
net.inet.tcp.maxtcptw = 47120
```
Benchmark (Case 1)

- Increased tcp-backlog setting to high value in the Redis config file
- http_load-14aug2014 (parallel and rate test)
- Tested URL/Domains:
  - http://fxr.watson.org/fxr/source/arm/lpc/lpc_dmac.c
  - http://mongolian-it.blogspot.com/
  - http://www.patrick-wied.at/static/nudejs/demo/
  - http://news.gogo.mn/
  - http://www.amazon.com/
  - http://www.uefa.com/
Benchmark (Case 1)

- http://www.tmall.com/
- http://www.reddit.com/r/aww.json
- http://nginx.com
- http://www.yahoo.com
- http://slashdot.org/?nobeta=1
- http://www.ikon.mn
- http://www.gutenberg.org
- http://www3.nd.edu/~dpettifo/tutorials/testBAD.html
- http://penthouse.com/#cover_new?{}
- http://www.playboy.com
- http://breakingtoonsluts.tumblr.com/
Benchmark (Case 1)

- Test commands used for HTTP load tests:
  
  ./http_load -proxy 172.16.2.1:8080 -parallel 10 -seconds 600 urls
  ./http_load -proxy 172.16.2.1:8080 -rate 10 -jitter -seconds 600 urls

- -parallel : number of concurrent connections to establish and maintain
- -rate : number of requests sent out per second
- -jitter : varies the rate by about 10%
- -seconds : number of seconds to run the test
## Benchmark (Case 1) results

<table>
<thead>
<tr>
<th>No</th>
<th>Result names</th>
<th>Parallel test</th>
<th>Rate test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Shuultuur</td>
<td>Dansguardian</td>
</tr>
<tr>
<td>1</td>
<td>Fetches</td>
<td>17654</td>
<td>4298</td>
</tr>
<tr>
<td>2</td>
<td>Max parallel</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Mean bytes/connection</td>
<td>79213.8</td>
<td>94820.7</td>
</tr>
<tr>
<td>4</td>
<td>Fetches/sec</td>
<td>29.4233</td>
<td>7.16333</td>
</tr>
<tr>
<td>5</td>
<td>Msecs/connect</td>
<td>0.189717 mean, 13.855 max, 0.088 min</td>
<td>0.184428 mean, 0.485 max, 0.088 min</td>
</tr>
<tr>
<td>6</td>
<td>Msecs/first-response</td>
<td>229.182 mean, 5114.55 max, 8.049 min</td>
<td>1374.9 mean, 40977.9 max, 0.779 min</td>
</tr>
<tr>
<td>7</td>
<td>Timeouts</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>Bad byte counts</td>
<td>6660</td>
<td>1415</td>
</tr>
<tr>
<td>9</td>
<td>HTTP response codes</td>
<td>200</td>
<td>12120</td>
</tr>
<tr>
<td>10</td>
<td>301</td>
<td>714</td>
<td>191</td>
</tr>
<tr>
<td>11</td>
<td>302</td>
<td>819</td>
<td>171</td>
</tr>
<tr>
<td>12</td>
<td>403</td>
<td>3843</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>404</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>500</td>
<td>148</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>503</td>
<td>-</td>
<td>341</td>
</tr>
</tbody>
</table>
Benchmark (Case 1) results

- Shuultuur has some advantages and disadvantages
  - Internal Server Error (500) more often than Dansguardian
  - More successful responses (200).
- Dansguardian
  - Responded 341 times with Service Unavailable (503)
  - Much more timeouts.
- On the performance side, in average, Shuultuur’s performance was higher than Dansguardian in most cases for both tests.
Benchmark (Case 2)

• Test environment (Case 2)
  • Server OS
    • FreeBSD 10.1-RELEASE amd64
  • Server hardware:
    • CPU – AMD G series T40E, 1 GHz dual Bobcat core with 64 bit support, 32K data + 32K instruction + 512K L2 cache per core
    • Memory - 4096MB
  • Go version 1.4.1
  • Squid and Dansguardian versions are same as before
Benchmark (Case 2)

- /etc/sysctl.conf includes following:
  
  kern.ipc.somaxconn = 4096
  kern.maxfiles = 10000
  kern.maxfilesperproc = 8500
  kern.ipc.maxsockets = 6500
  kern.ipc.nmbclusters = 20000
  net.inet.tcp.maxtctw = 4000

- Changed tcp-backlog setting to 4096 in the Redis config file
- http_load-03feb2015 (parallel and rate test)
# Benchmark (Case 2) results

<table>
<thead>
<tr>
<th>No</th>
<th>Result names</th>
<th>Parallel test</th>
<th>Rate test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Shuultuur</td>
<td>Dansguardian</td>
</tr>
<tr>
<td>1</td>
<td>Fetches</td>
<td>4319</td>
<td>2643</td>
</tr>
<tr>
<td>2</td>
<td>Max parallel</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Mean bytes/connection</td>
<td>120364</td>
<td>134945</td>
</tr>
<tr>
<td>4</td>
<td>Fetches/sec</td>
<td>7.19813</td>
<td>4.405</td>
</tr>
<tr>
<td>5</td>
<td>Msecs/connect</td>
<td>19.193 mean, 3009.89 max, 0.925 min</td>
<td>6.23727 mean, 53.385 max, 0.991 min</td>
</tr>
<tr>
<td>6</td>
<td>Msecs/first-response</td>
<td>764.861 mean, 59830.3 max, 36.664 min</td>
<td>1337.36 mean, 55849.5 max, 16.704 min</td>
</tr>
<tr>
<td>7</td>
<td>Timeouts</td>
<td>28</td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>Bad byte counts</td>
<td>1787</td>
<td>2160</td>
</tr>
<tr>
<td>9</td>
<td>HTTP response codes</td>
<td>200</td>
<td>3677</td>
</tr>
<tr>
<td>10</td>
<td>301</td>
<td>9</td>
<td>191</td>
</tr>
<tr>
<td>11</td>
<td>302</td>
<td>366</td>
<td>217</td>
</tr>
<tr>
<td>12</td>
<td>403</td>
<td>233</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>404</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>500</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>503</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Benchmark (Case 2) results

- Shuultuur’s performance was higher than Dansguardian in most cases for both tests
- System load average especially CPU usage was high when Shuultuur was working
Benchmark (Case 2) results

- top report when running Shuultuur:

lastpid: 1317; load averages: 1.52, 1.00, 0.58
71 processes: 1 running, 64 sleeping, 6 stopped
CPU: 31.4% user, 0.0% nice, 5.9% system, 1.6% interrupt, 61.2% idle
Mem: 58M Active, 189M Inact, 158M Wired, 70M Buf, 3519M Free
Swap: 978M Total, 978M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>user</td>
<td>18</td>
<td>25</td>
<td>0</td>
<td>84540K</td>
<td>43672K</td>
<td>uwait</td>
<td>1</td>
<td>6:16</td>
<td>91.85%</td>
<td>shuultuur</td>
</tr>
<tr>
<td>1299</td>
<td>user</td>
<td>5</td>
<td>21</td>
<td>0</td>
<td>28544K</td>
<td>9484K</td>
<td>piperd</td>
<td>1</td>
<td>0:18</td>
<td>4.10%</td>
<td>gcvis</td>
</tr>
<tr>
<td>822</td>
<td>redis</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>28108K</td>
<td>6540K</td>
<td>uwait</td>
<td>1</td>
<td>0:21</td>
<td>0.29%</td>
<td>redis-server</td>
</tr>
<tr>
<td>1024</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>17092K</td>
<td>select</td>
<td>0</td>
<td>3:42</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>794</td>
<td>squid</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>164M</td>
<td>68400K</td>
<td>kqread</td>
<td>1</td>
<td>1:20</td>
<td>0.00%</td>
<td>squid</td>
</tr>
<tr>
<td>1030</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18660K</td>
<td>select</td>
<td>1</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1028</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18664K</td>
<td>select</td>
<td>1</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1029</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18672K</td>
<td>select</td>
<td>1</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1033</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18664K</td>
<td>select</td>
<td>0</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1032</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18660K</td>
<td>select</td>
<td>0</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1031</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18672K</td>
<td>select</td>
<td>1</td>
<td>0:02</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
</tbody>
</table>
Benchmark (Case 2) results

- Dansguardian:

  lastpid: 1151; load averages: 0.42, 0.68, 0.81
  156 processes: 1 running, 152 sleeping, 3 stopped
  CPU: 0.2% user, 0.0% nice, 10.2% system, 1.8% interrupt, 87.8% idle
  Mem: 103M Active, 245M Inact, 161M Wired, 58M Buf, 3415M Free
  Swap: 978M Total, 978M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>root</td>
<td>1</td>
<td>35</td>
<td>0</td>
<td>43580K</td>
<td>17092K</td>
<td>nanslp</td>
<td>0</td>
<td>1:13</td>
<td>23.49%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>794</td>
<td>squid</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>160M</td>
<td>62060K</td>
<td>kqread</td>
<td>0</td>
<td>0:13</td>
<td>4.59%</td>
<td>squid</td>
</tr>
<tr>
<td>1002</td>
<td>user</td>
<td>19</td>
<td>42</td>
<td>0</td>
<td>93636K</td>
<td>51320K</td>
<td>STOP</td>
<td>0</td>
<td>9:58</td>
<td>0.00%</td>
<td>shuultuur</td>
</tr>
<tr>
<td>1001</td>
<td>user</td>
<td>6</td>
<td>20</td>
<td>0</td>
<td>33856K</td>
<td>10692K</td>
<td>STOP</td>
<td>0</td>
<td>0:32</td>
<td>0.00%</td>
<td>gcvis</td>
</tr>
<tr>
<td>822</td>
<td>redis</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>28108K</td>
<td>6452K</td>
<td>uwait</td>
<td>1</td>
<td>0:15</td>
<td>0.00%</td>
<td>redis-server</td>
</tr>
<tr>
<td>932</td>
<td>user</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>21916K</td>
<td>3244K</td>
<td>CPU0</td>
<td>0</td>
<td>0:06</td>
<td>0.00%</td>
<td>top</td>
</tr>
<tr>
<td>1028</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18152K</td>
<td>select</td>
<td>0</td>
<td>0:01</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1033</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18172K</td>
<td>select</td>
<td>0</td>
<td>0:01</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>926</td>
<td>user</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>86472K</td>
<td>7240K</td>
<td>select</td>
<td>1</td>
<td>0:01</td>
<td>0.00%</td>
<td>sshd</td>
</tr>
<tr>
<td>1025</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>31292K</td>
<td>5328K</td>
<td>select</td>
<td>1</td>
<td>0:00</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1030</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18304K</td>
<td>select</td>
<td>0</td>
<td>0:00</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
<tr>
<td>1053</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>43580K</td>
<td>18664K</td>
<td>select</td>
<td>0</td>
<td>0:00</td>
<td>0.00%</td>
<td>dansguardian</td>
</tr>
</tbody>
</table>
Conclusions and future works

• Developing application in Go is simple
  • Using built-in data structures such as maps and slices
  • Many open source projects were useful
• http_load test was run multiple times and results were consistent
• Results will be lot better when we solve problems
Conclusions and future works

• Lack of fast and stable image checking feature
• High number of goroutines problem when load is high
  • Use channels for incoming requests to have some queuing mechanism
• Last but not least
  • The memory usage and CPU load problem is a major issue for embedded system applications
  • Planning to do more research on this to stabilize the resource usages.
• Any comments and ideas related to Shuultuur
  • Contact: ganbold@gmail.com
Thank you for your attention

Questions?