CONCURRENCY OUTLOOK USING CMS DAS SERVICE

Valentin Kuznetsov
Cornell University

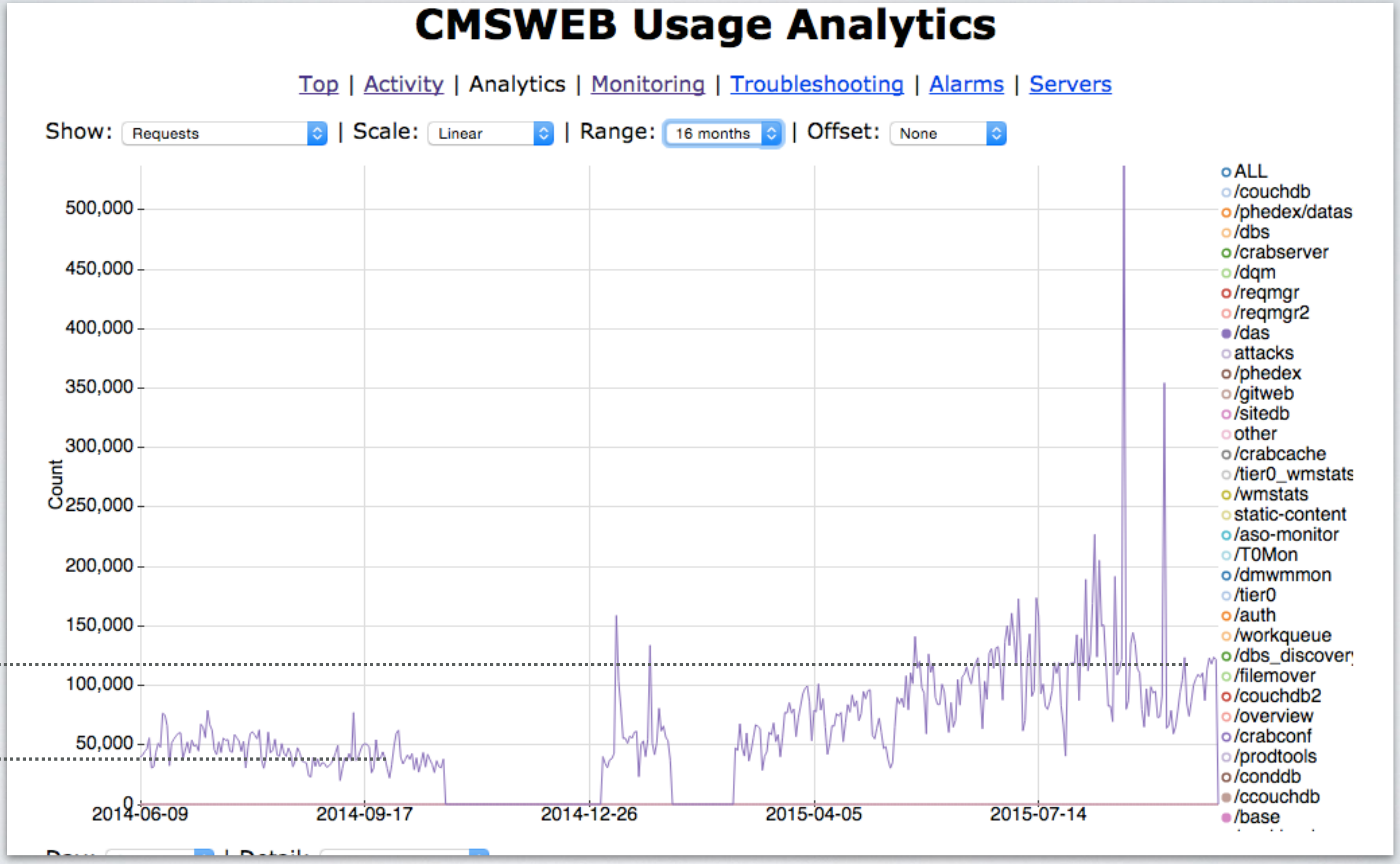
PREFACE

- Did you ever try to switch from traditional hard-drive to SSD?
- This talk is not about new language per-se and neither any ads, promotions or requests to use it, but feel free to make your own conclusions.
- This was my hobby project done in spare time to learn new ideas, technique, technologies about concurrency
- Compare Apples to Apples using CMS DAS service, i.e. write code for real application in two languages and benchmark it on real user queries using different technologies and same application design
    - the outcome speaks for itself and worth to share

DAS USE CASE

- DAS server needs to handle concurrency really well:
  - support complex look-up queries and concurrent clients
  - single user query can resolve into N data-provider calls, currently we are in 1:10 - 1:100 regime
  - single request requires multiple APIs, URLs, parsers and perform work in parallel
  - data are stored into raw/merge cache 10k docs/second
  - data retrieved from merge cache
- DAS client uses retry approach, place request and wait for results via periodic check of request id

DAS STATISTICS



2.5-3x

PROBLEM

- We expect the load will ramp up in coming years
    - queries can become more complex size wise/number of calls to data-providers, we are going to have more data, didn't we?
- DAS requires and use concurrent design, but
- Current Python implementation still depends on GIL
- We can scale horizontally but it does not resolve issues with burst of concurrent requests, neither allow to utilize all CPU on a server
- Firing up Python subprocesses (or use multiprocessing module) does not solve the problem due to system limitations, sharing data-structures, connection pools, etc. I tried many ideas without luck of further improvements within current tool set.

POSSIBLE SOLUTION

- Switch to modern programming language with built-in concurrency primitives
- Go-language, developed by Google in 2007
  - statically typed, syntax loosely derived from C, garbage collection, type safety, message passing via goroutines/channels (similar to lightweight Erlang processes)
  - code syntax is similar to python and C, way to easy to program than Erlang, provides GC, lightweight goroutines (a-la unix &)

APPLES TO APPLES

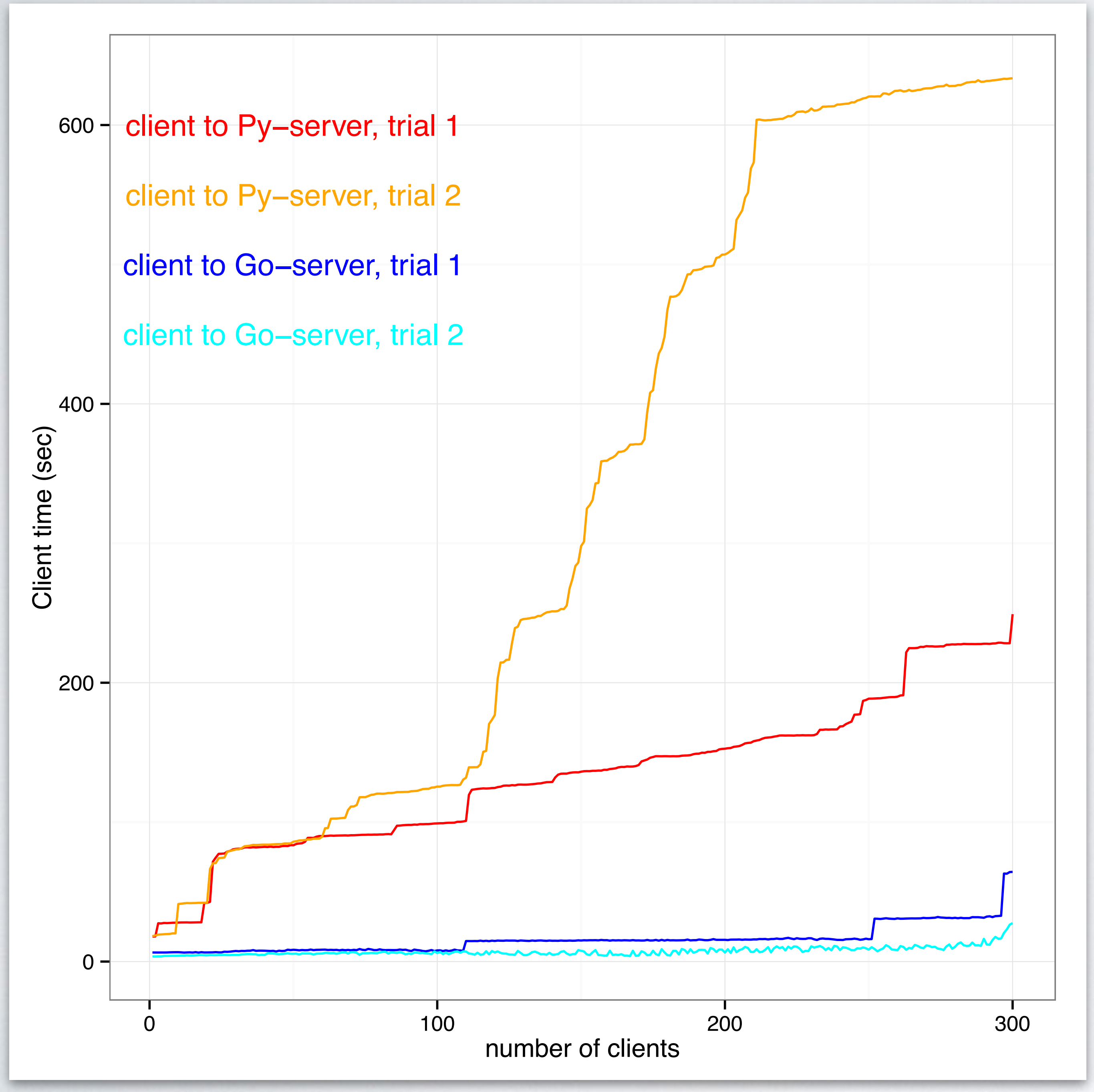
- DAS python server: 26.5k lines of code plus third-party software (CherryPy, pycurl, Cheetah)
- Use pymongo driver.
- 3 thread pools, 175 threads, O(min) to start-up
- Parallel execution via thread pool

- DAS Go server: 4k lines of code (built-in template/web systems). Compile time 2 sec
- Use mgo driver
- No threads in code, at run time 15 threads + goroutines handling the requests load, zero startup time
- Parallel execution via **go func()**

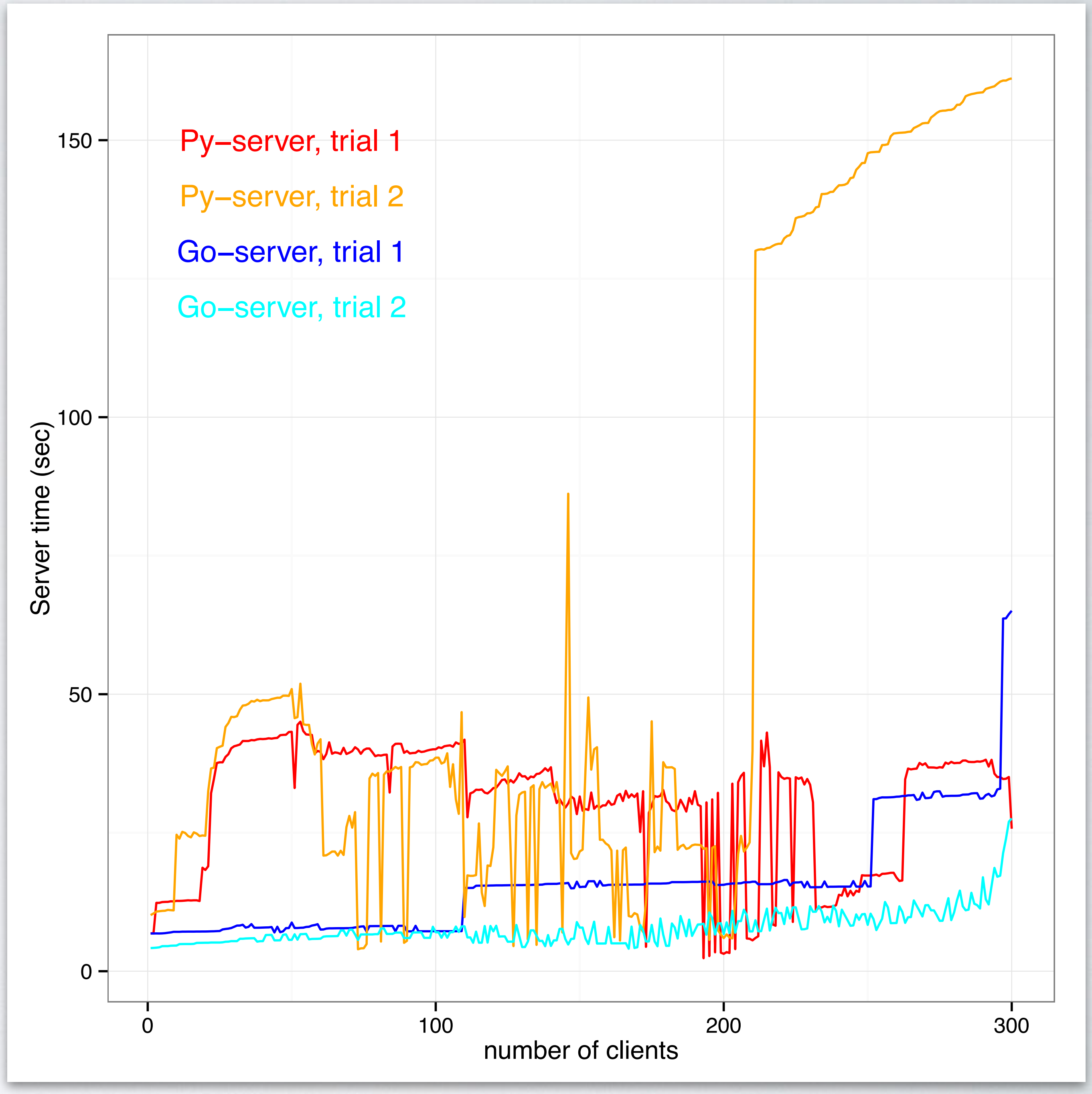
Use the same DAS python client to talk to both services
To simulate real clients we use python multiprocessing functionality
All tests were done on MacBookAir Intel Core i7, 8GB of RAM.

PERFORMANCE

Client time



Server time

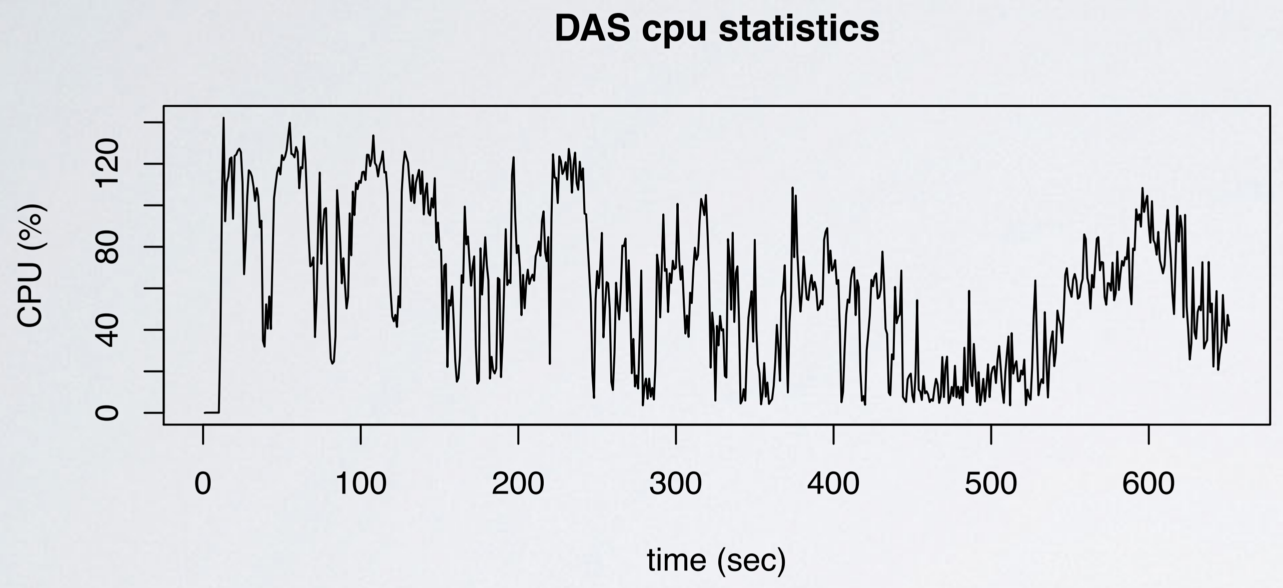


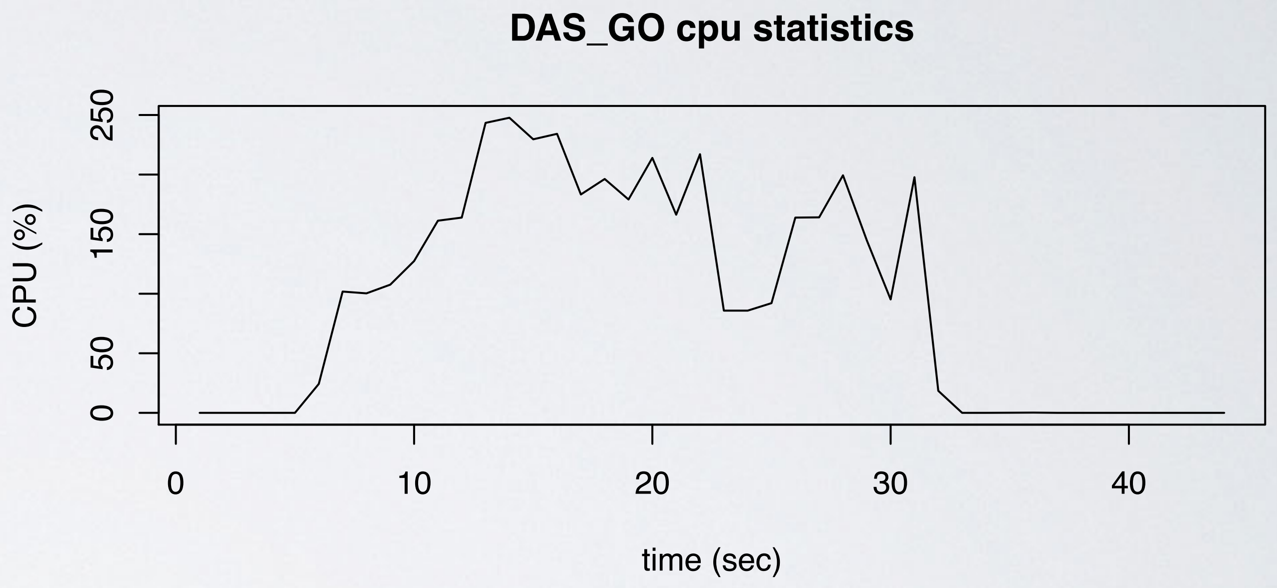
file dataset=/A/B/C queries, 300 clients, 300 queries => 80k files

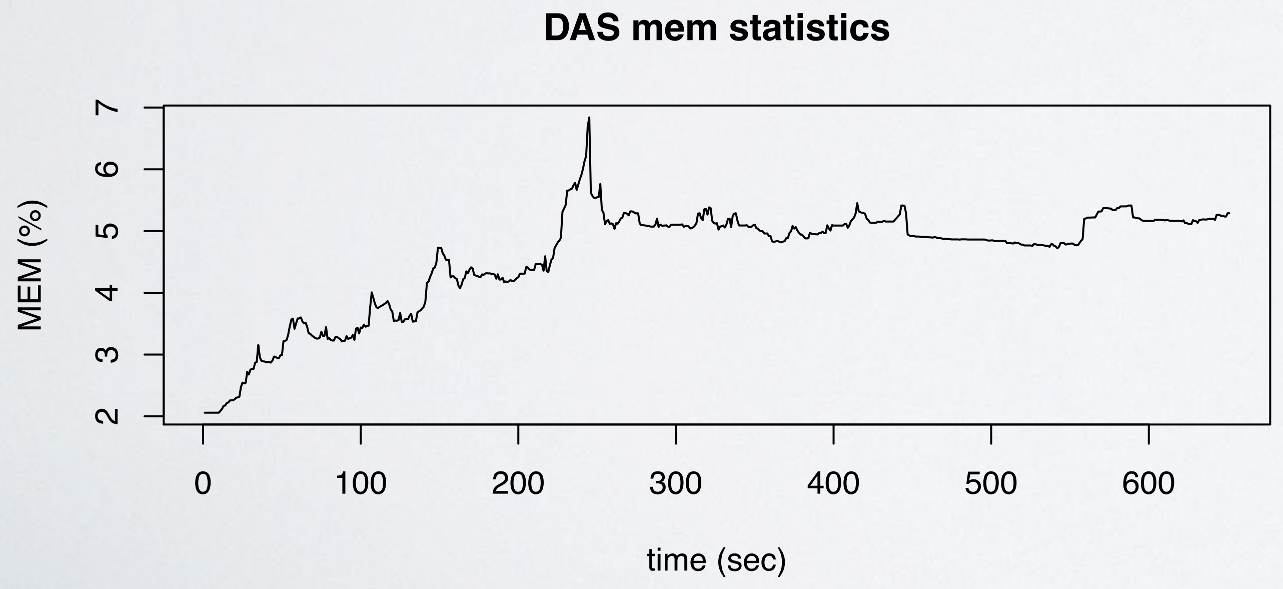
CPU & RAM

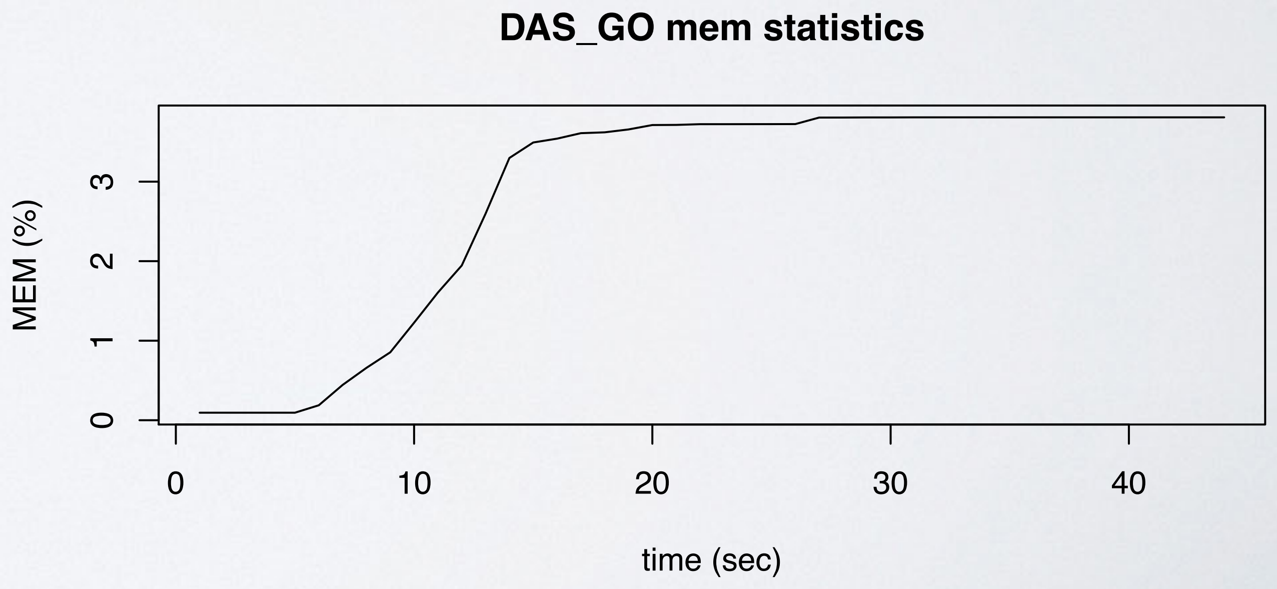
DAS Py-server

DAS Go-server

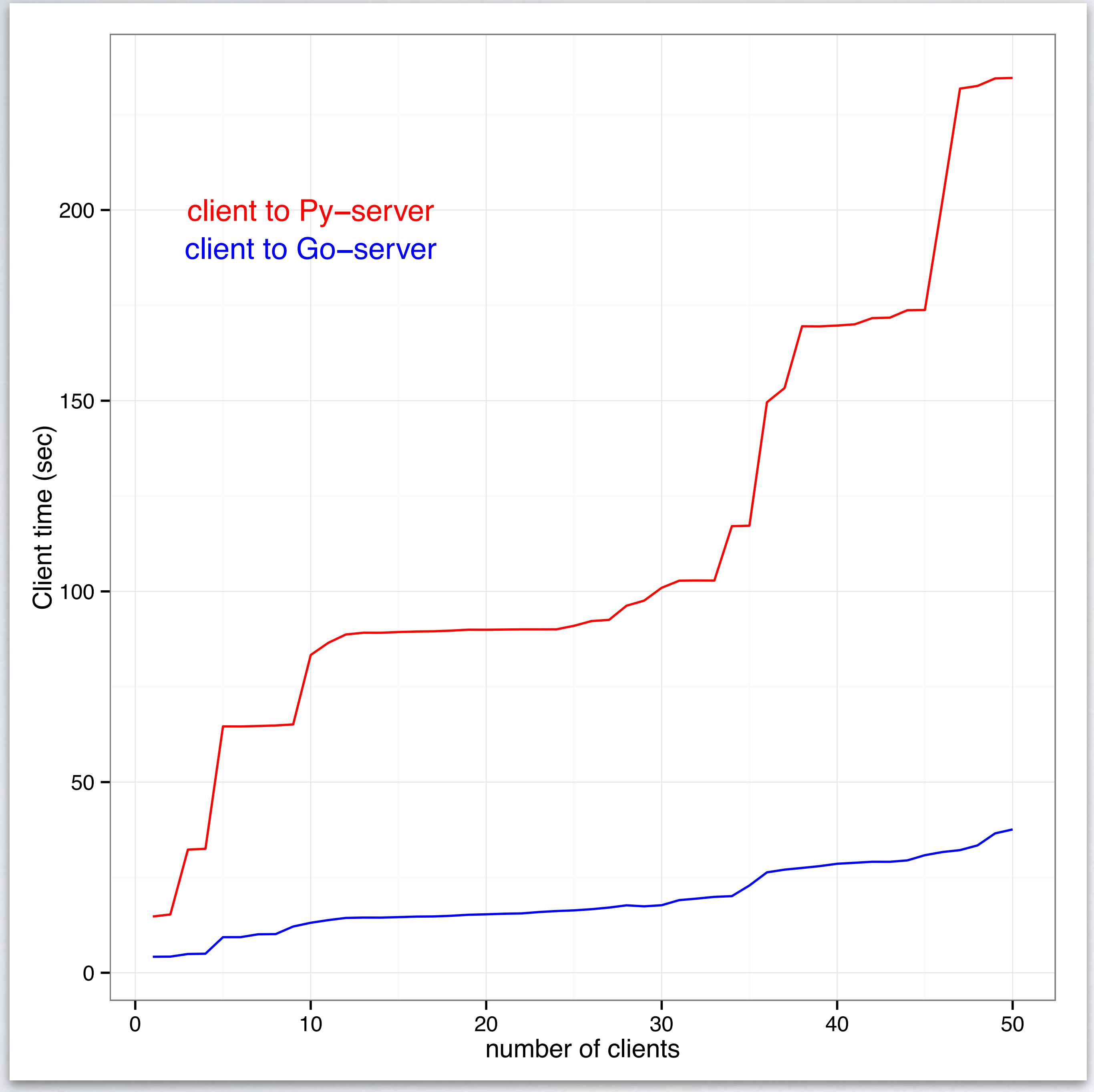








CMS SERVICES ISSUE



file, run, lumi dataset=/a/b/c
50 requests => 37k records

- DAS Go server can easily saturate CMS data-services
- When requested queries require complex look-up, CMS services fail to cope with load
- file, run, lumi dataset=/a/b/c look-up requires to get all blocks for dataset and query individual block to get file, run, lumi information
- 50 parallel queries resolves into 5327 requests burst and I saw many failures on DBS side
- Therefore re-writing DAS may not solve all issues

GO CODE

- Easy to write, syntax somewhat similar to python and C/C++, e.g.

```
import “log”
func PrintArray(arr []string) {
  for idx, val := range arr {
      log.Println(idx, val) // Print index and value of array element
  }
}
go PrintArray(someArray) // will execute code in parallel, a-la shell# cmd &
```

- Reach standard library, e.g. no need to write web server, thread pool, processes
- Built-in template support, Django syntax
- Built-in support to import from github, e.g. import “github.com/user/bla”
- go doc, go fmt, go get, go run, go build
- FAST, REALLY FAST compile time into static executable. Entire DAS server fits in 10MB executable file and takes only 2 seconds to compile.

SUMMARY

- Hobby project shows real potential for server side application (DAS server):
  - Code reduction: 6 times
  - Memory reduction: 1.5-2 times
  - Run and scale on all CPUs natively, concurrency part of the language
  - Client elapsed time decrease 10 times or more
  - Learn Go syntax: 1 hour, reaching master level will vary on your ability
- Current DAS server is limited by GIL, hard to handle more than 300 clients at once. Go DAS server can easily scale to larger load, tested with 1000 clients on a single node.

REFERENCES

- Go language
  - http://golang.org
- CSP model:
  - http://en.wikipedia.org/wiki/Communicating_sequential_processes
- R. Pike, Concurrency Is Not Parallelism
  - https://vimeo.com/49718712
  - http://concur.rspace.googlecode.com/hg/talk/concur.html
- R. Pike, Concurrency Patterns:
  - http://talks.golang.org/2012/concurrency.slide