Concurrency studies

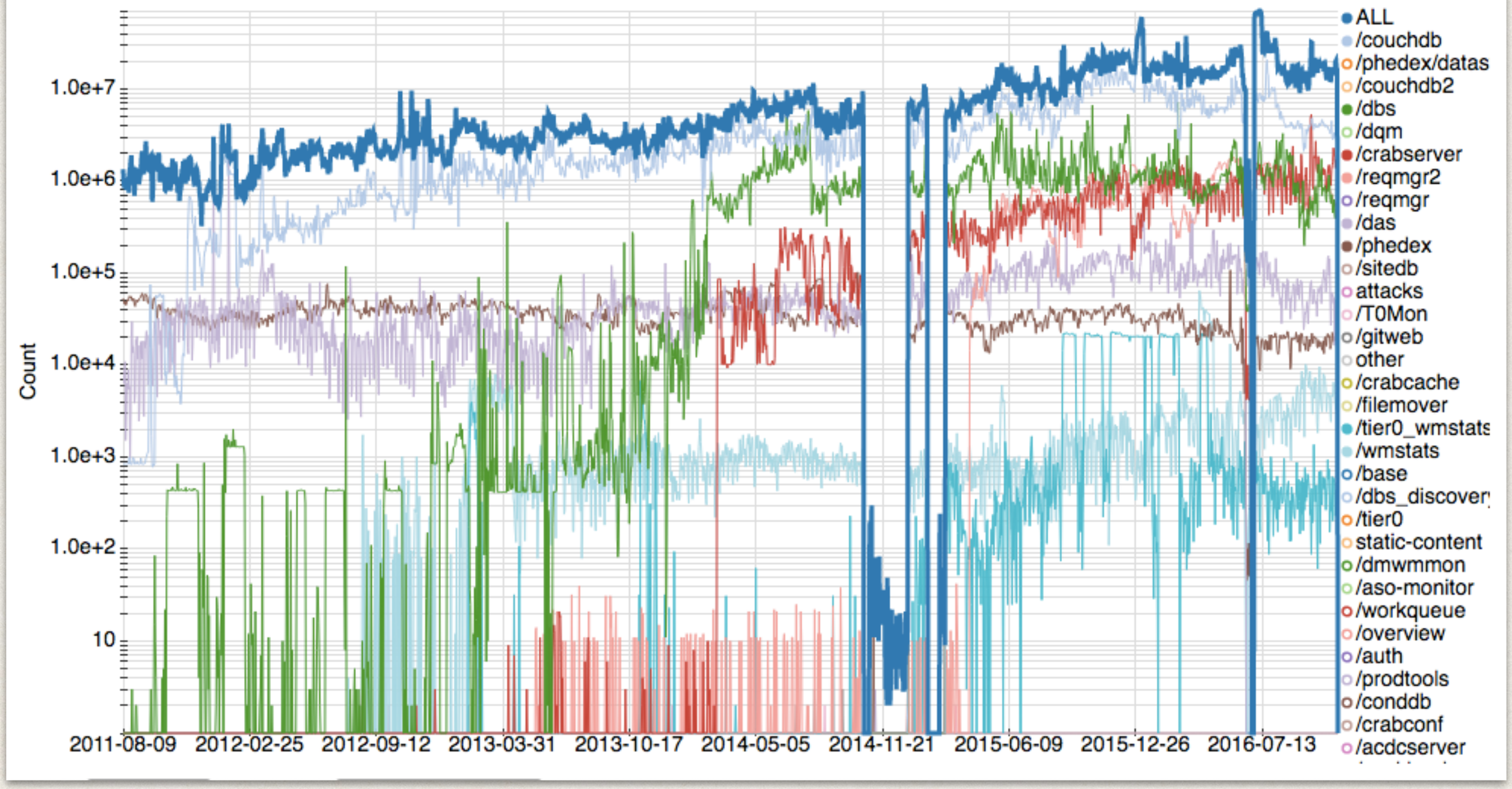
Valentin Kuznetsov, Cornell University

CMS DMWM meeting

CMSWEB Usage Analytics

Top | Activity | Analytics | Monitoring | Troubleshooting | Alarms | Servers

Show: Requests | Scale: Logarithmic | Range: 64 months | Offset: None



T0Wmstats WMStats PhEDEx DAS CRABServer DBS CouchDB All

low → high

Statement of work

- Compare various technology solutions for next generation of CMS web applications
  - Python based solutions (CherrPy vs Flask vs Flask+uWSGI) vs Go-based one
- Understand effect of cmsweb authentication
- Benchmarks:
  - pure framework throughput by measuring load against HTTP server w/ static content
  - cmsweb effect by measuring load of frontend+HTTP backend
  - DMWM stack effect by measuring static DB content
  - DB effect by measuring heavy DB “random” queries

New technology choices

- Flask micro-framework is a event based web framework based on Werkzeug WSGI utility library and greenlets
  - The greenlets are micro-threads called “tasklets” which run in a single thread and provide *pseudo-concurrent* capabilities in Python (coroutines vs system threads)
    - the concept is quite similar to Go language, but still limited to single Python thread (to avoid GIL), its scalability can be achieved by running multiple Flask processes behind WSGI
    - it may outperform thread-based web frameworks (like CherryPy) for high concurrent applications
    - Recently PdmV/McM group switched to it instead of DMWM/CherryPy stack
- Go is a compiled, statically typed language with garbage collection and builtin concurrency support via Communicating Sequential Processes (CSP). The web tools (web server, templates, etc.) are part of its standard library.

Tests environment/notations

- We used 2 backend and 2 frontend nodes (SLC7 and SLC6), each node has 4 cores and 8GB of RAM
- **Static content** tests was performed against code which served "Hello World" string and nothing else
- **Static query** tests was performed against DBS datasets API where we pass /Zee*/*/USER query
- **Random query** tests was performed against DBS files, blocks, datasets APIs where we pass AOD datasets paths as input parameter
- **DBS+Flask** code is simplified version of DBSReaderModel class integrated into Flask application stack
- **DBS+Flask+uWSGI** stack used uWSGI server with 4 processes, 100 threads
- **Go+16 cores** tests run outside of cmsweb-testbed cluster on SLC7 node with 16 cores and 28 GB of RAM
- **Go+https** means that Go server serves HTTPs requests with own auth module checking user DN against pre-fetched SiteDB
- **dbs2go** is full implementation of DBS server read APIs in Go language
- **Go+frontend, dbs+frontend** means that Go server serves HTTP requests behind cmsweb apache frontends
- **Hey** Go tool used to simulate the load, e.g. 5k/200 (5000 requests with 200 concurrent clients)

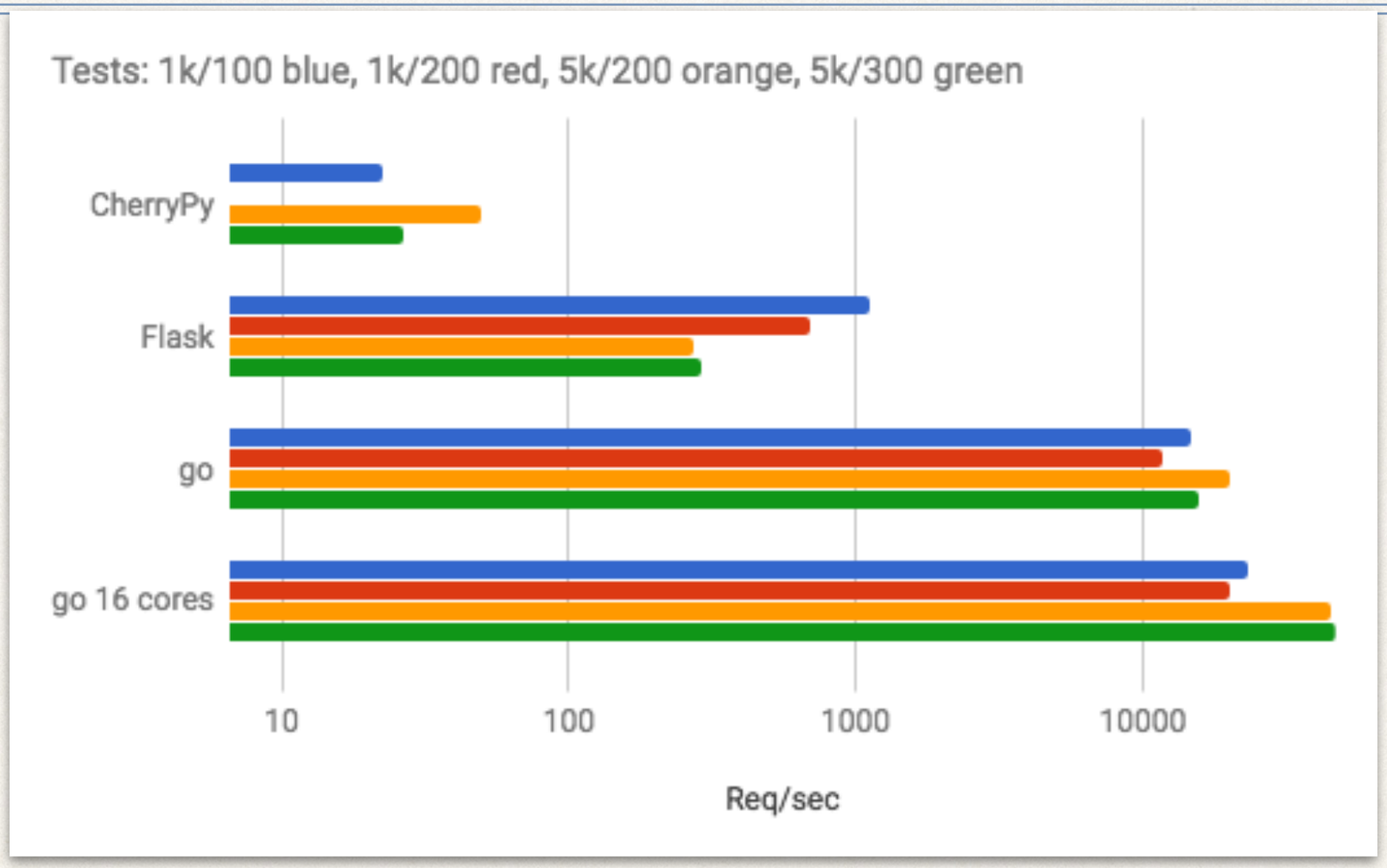
Hello world (static content)

```
package main
import "net/http"
func defaultHandler(w http.ResponseWriter, r *http.Request) {
        w.WriteHeader(http.StatusOK)
        w.Write([]byte("Hello from Go"))
}
func main() {
        http.HandleFunc("/", defaultHandler)
        server.ListenAndServe()
    }
```

```
import cherrypy
class Main(object):
    def index(self):
        return "Hello from CherryPy”
    index.exposed=True
cherrypy.root = Main()
if __name__ == '__main__':
   cherrypy.quickstart(Main(), '/')
```

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "Hello from Flask"
if __name__ == "__main__":
    app.run()
```

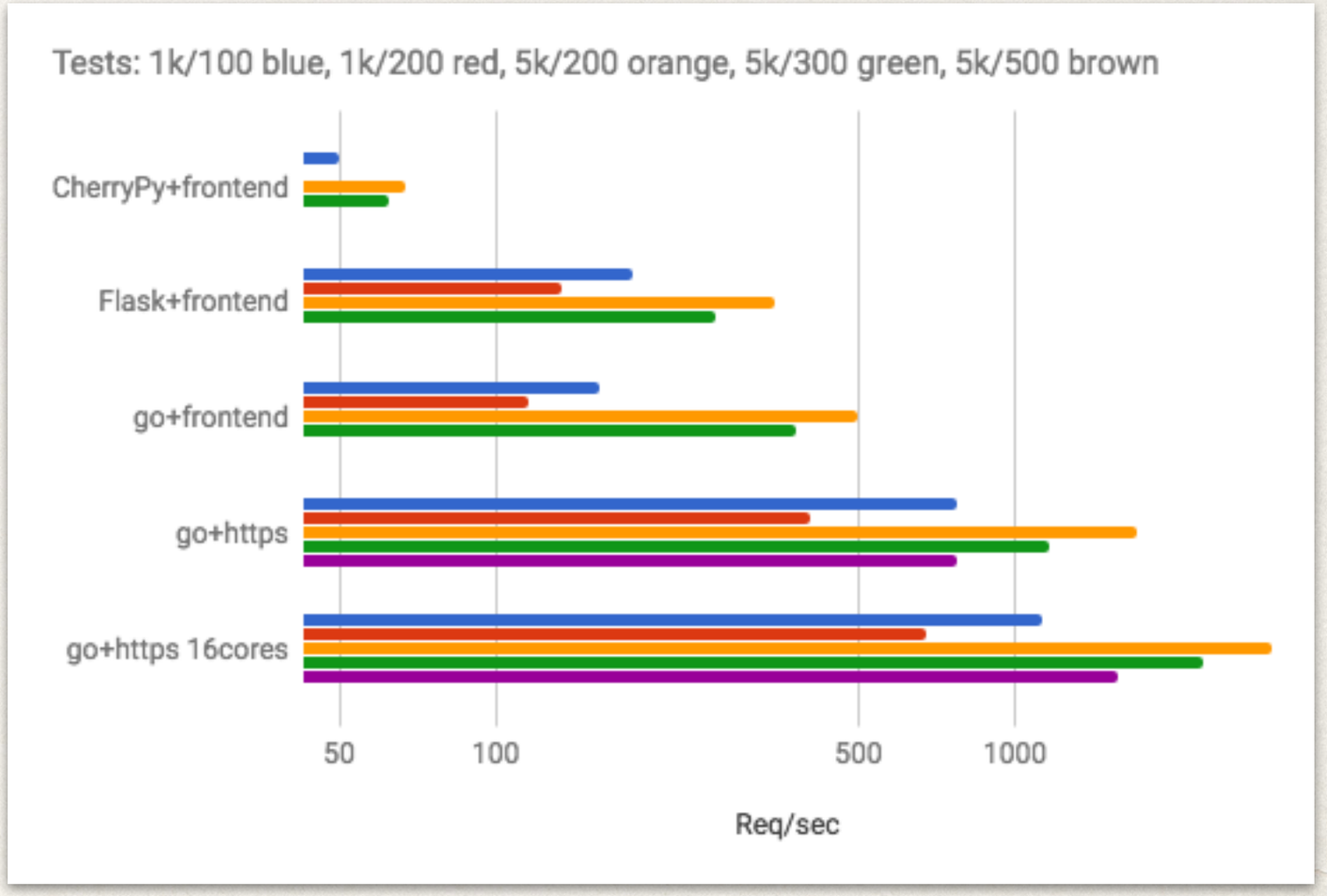
Comparison of HTTP frameworks



- This test shows ultimate performance of individual frameworks
  - Go >> Flask > CherryPy
  - Go HTTP server is part of Standard Library, both CherryPy and Flask are third-party packages for Python

Effect of cmsweb (apache)

- CherryPy+frontend is our benchmark, ultimate performance of our CMS data services
- cmsweb frontend effect is visible:
  - improves throughput of python based app by sharing HTTP connections
  - significantly decrease throughput of Go server
- Python/apache died for tests with more than 5k/300 requests
- Go server performed well on all tested and can scale beyond (was tested with 5k/1000 load), scale with number of cores



log scale

First round of conclusions

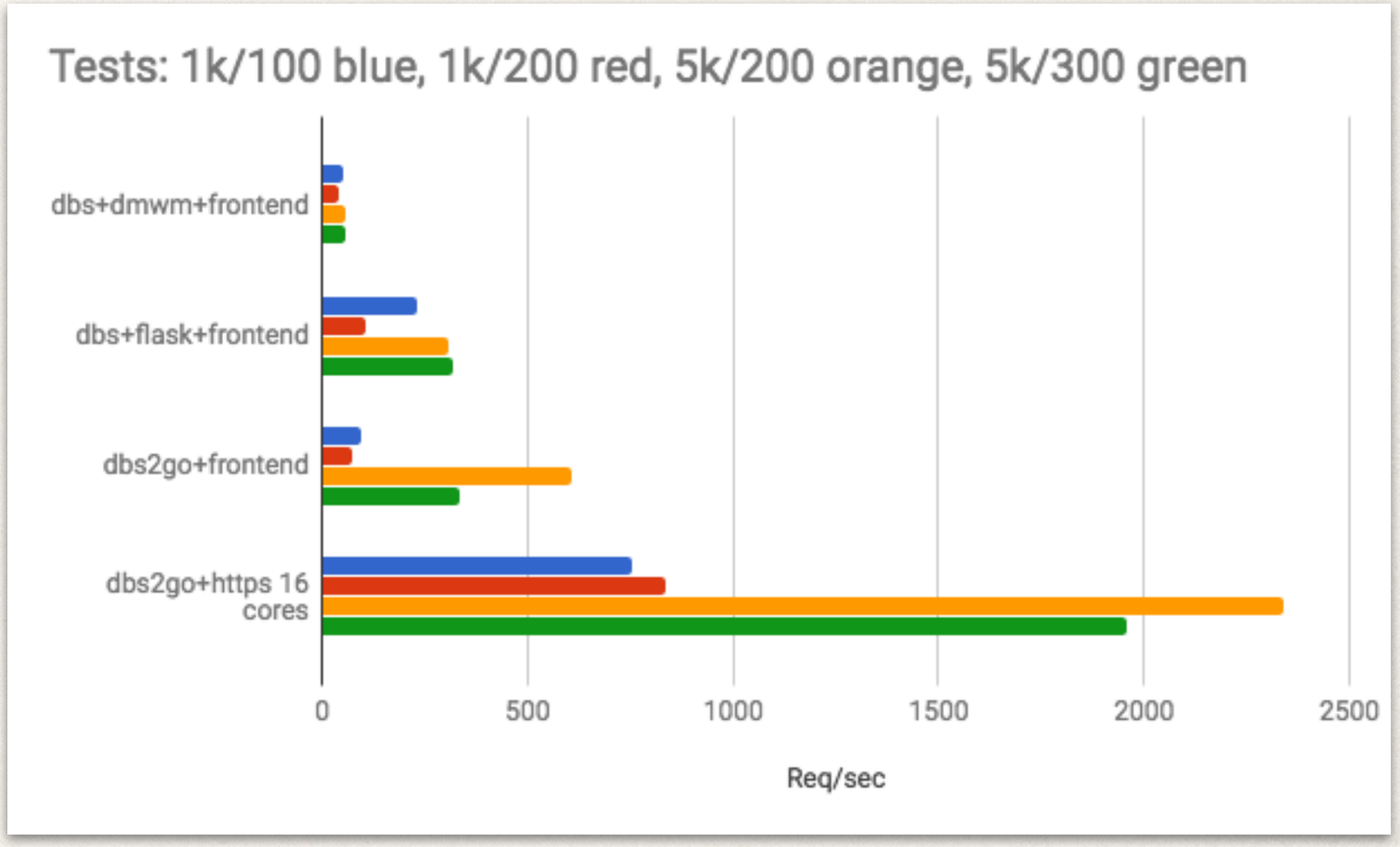
	CherryPy	Flask	Go	Go+ 16 cores	CherryPy+ frontend	Flask+ frontend	Go+ frontend	Go+https	Go+https 16 cores
1k/100	22	1121	14613	23301	50	182	159	773	1131
1k/200	7	692	11605	20190	42	134	115	403	677
5k/200	49	273	20211	45252	67	346	498	1717	3134
5k/300	26	287	15802	46896	62	265	377	1171	2307
5k/500				38945				776	1578
5k/1000				4629				290	661

*Table shows **Req/second** for different tests*

- Go >> Flask >> CherryPy serving static content
- Go > Flask > CherryPy via frontends
- Go+HTTP >> Go+HTTPs > Go+frontend
- apache helps python frameworks, but hurts throughput of Go

Tests with DBS static query

- DBS+DMWM+frontend is our benchmark
- I hacked DBS code to use it within Flask framework
- Use dbs2go with oci8 ORACLE driver for
  - HTTP server behind frontend
  - HTTPs server w/o frontend
- **All tests used dataset pattern query /Zee*/*/USER**



DBS+DMWM, DBS+Flask and dbs2go +frontend tests run on 2 back-ends (SLC6, SLC7) and 2 frontends. Each node has 4 cores with 8GB of RAM.

dbs2go+https runs on SLC7 16 cores node with 28 GB of RAM.

Tests with DBS random queries

	DBS static	DBS random	DBS+Flask static	DBS+Flask random	DBS+Flask +uWSGI random	Go+fr-end static	Go+fr-end random	Go+https 16 cores static	Go+https 16 cores random
1k/100	43	48	157	3.8	63	95	157	751	564
1k/200	19	44	130	1	67	74	155	834	599
5k/200	57	51	305	0.2	79	608	549	2339	1630
5k/300	54	52	319	0.3	79	335	331	1958	1386

*Table shows **Req/second** for different tests*

- **Random query** tests: I used AOD datasets, created file, block, dataset queries from them and used them in tests
- Flask fails miserably in random tests due to many factors:
  - it does not share DB connections and current implementation of DBS is multi-threaded. The fix will require DMWM code re-factoring, remove multi-threading conditions, etc.
  - its throughput can be recovered by using uWSGI (Web Server Gateway Interface) to run multiple (Flask) applications
- Go is free from all of these limitations, apache frontend significantly slows down its throughput

Final standing in random tests

- Go >> DBS+Flask+uWSGI > DBS+DMWM
- Flask behind uWSGI performs 30% better than DBS+DMWM
  - uWSGI server runs multiprocess (Flask) app to scale with number of cores

Summary

- Current system based on CherryPy and DMWM framework is decade old
- Modern Python web framework (Flask) may provide significant boost of performance by avoiding system threads context switching (greenlets, coroutines within single thread). In order to use them code should be structured appropriately and more R&D is required to understand its suitability for DMWM.
- Apache front-ends
  - increase throughput of python based data-services (via shared HTTP connection pool),
  - significantly limit throughput of highly concurrent HTTP servers
  - apache cmsweb dies at high concurrent load (can't sustain more than 500 concurrent clients)
- Go solution
  - throughput of Go >> Flask > CherryPy, plus it scales with number of cores
  - throughput of HTTP Go >> HTTPs Go >= dbs2go (querying ORACLE DB)
  - throughput of HTTPs Go server outperform apache+python solution by large factor