# Core, Framework & Externals

Adrià Casajús & Zoltan Mathe

University of Ferrara

20150528

## Attack plan

- The good
  - New error mechanism
  - Component profiling/monitoring
  - Connection retry
  - RFC proxies
- The bad
  - Externals
    - The very bad: openssl
- The nice
  - Future monitoring and NoSQL profiling

## New error mechanism

- The good ol’ S_OK/S_ERROR has some drawbacks:
  - Programmatically difficult to react depending on errors
  - Bad traceability (where was the error generated?)
- Could be improved by using exceptions instead of S_*
  - At the beginning it was decided we weren't going to use them
  - Adding now exceptions is really painful (try/catch everywhere)

## Error handling improvement

- Add a numeric value with semantic meaning to errors
  - Allows devs to react to different types of errors easily
  - Less typo prone, case insensitive

- Include the stack-trace of the error creation point in the error itself

- BUT replacing S_* is a pain in the a**
  - Has to be backwards compatible

# Enter DError

```
return DError( ENOENT, "File {} does not exist".format( fileName ) )
```

- Includes error number and originating callstack
  - print of the error will show the stack directly
- There's a method to check error type that's backwards compatible with S_ERROR
- Will land in somewhere in the next releases
- For more info Chris Haen is the person to nag :)

## System Administration

- Currently shows running components in each host of a an installation
  - Act on components/hosts
- Increasing number of hosts and DIRAC components
- Keeping track manually of big installations is an increasing problem

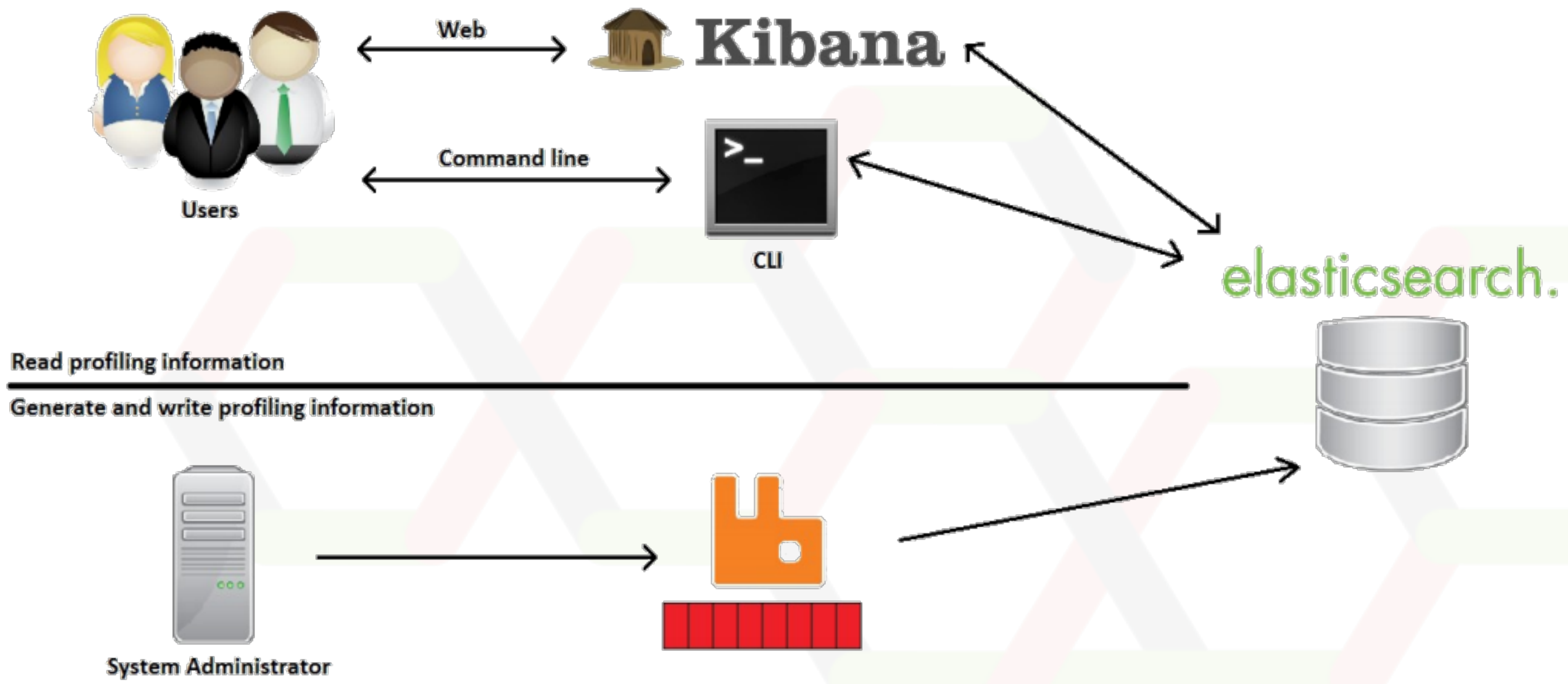
## Static component monitoring

- Keep track of
  - What was installed removed on each host
  - When was it done
  - Who did it
- In sync with actions taken from the system administrator
- Also keep track of non-DIRAC components via extensions
  - RabbitMQ, squid, vcycle…
- Will land in v6r13

## Dynamic component monitoring

- Profiling information of hosts and components
  - Metrics from non-DIRAC components also
- Updated regularly
- Host status stored in MySQL
  - Faster than query system administrator
- Profiling info is stored in ElasticSearch via RabbitMQ consumers

# Dynamic component monitoring


Web
Kibana
Users
Command line
CLI
elasticsearch.
Read profiling information
Generate and write profiling information
System Administrator

- For more info talk to Sergio Balbuena or to Federico

## Connection retries

- Currently if there's a connection error the client returns an error

- Implemented a connection retry BEFORE ANY DATA HAS ACTUALLY BEEN TRANSMITTED
  - Could delay a bit initialization of execution if configuration server is down but small price to pay compared with auto retries

- Ask Zoltan for more details

## RFC proxies

- Up until now we’ve been using “grid” proxies
  - First implementation of certificate proxies
  - Not standard outside WLCG/EMI/UMD/gLite
- A standardized format for proxies was created later
  - RFC3820 → RFC proxies
  - ☹ Require ASN.1 (de)serialization (Check out PR2272)
  - ☺ OpenSSL supports them!
- Everyone is moving towards using RFC proxies since some time ago

## DIRAC & RFC proxies

- DIRAC supports now RFC proxies
  - (well, starting from v6r14 I guess…)
  - Requires new version of pyGSI
    - Already included in the newest externals
- Since RFC proxies require decoding ASN.1 data DIRAC now can decode ASN.1 DER encoded data:
  - We can read VOMS extensions natively!
    - NO need for voms-proxy-info
  - We can't generate VOMS extensions
    - STILL NEED voms-proxy-init

## The bad
## (aka Externals)

## Externals

- A pain to maintain
- But we require pyGSI so they are needed
- pyGSI requires OpenSSL

- I f***ng hate OpenSSL
  - I invite anyone to have a walk amongst OpenSSL code
    - Like someone punching you in the eyes and kicking your brain at the same time

## OpenSSL security issues

- One of the most heavily used crypto/tls toolkits around

- Lots security issues
  - Heartbleed, POODLE, plenty of TLS errors/DoS, Mitm…
  - https://www.openssl.org/news/vulnerabilities.html
  - This is good

- Require using new versions of OpenSSL continuously

# BUT

- OpenSSL also changes stuff between versions
- Some nasty ones that exploded on us:
  - Changed requirements when decoding CSRs
    - usigned ones failed miserably
  - Changed some TLS config that prevents connecting to CASTOR2/DPM SRMs
    - But dCache/STORM is OK (:?)
- Also other toolkits change and they blame us that we can’t connect
  - Java 1.7 (I think) only allows a subset of EC ciphers and OSSL wanted to use parameters out of the allowed ones by Java

## OpenSSL

- As you can imagine dealing with this stuff is reeeeeally fun

- There's no easy alternative
  - Changing OpenSSL to GnuTLS or something like that requires rewriting pyGSI/DIRAC sec code

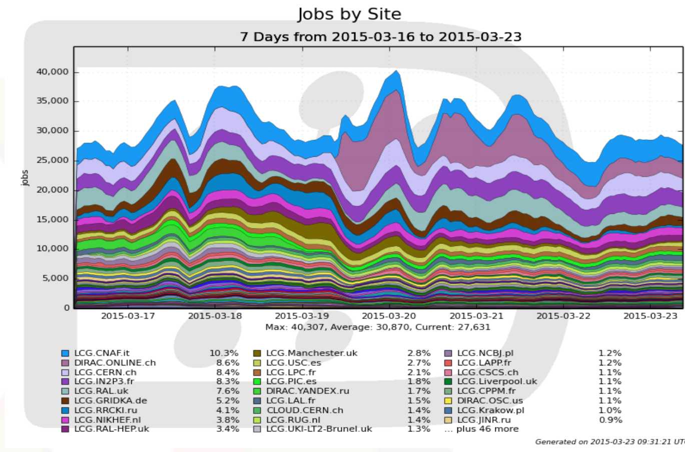
- We're stuck with it

- Somebody should start looking into pyGSI/OpenSSL code as I may not have much time in the near future

## The nice
## (aka NoSQL monitoring profiling)

## Motivation

- Develop a system for real time monitoring
- Why?
  - Current monitoring system (Accounting)
    - Not designed for real time monitoring
    - Hard to scale to hundred million records
- Goals:
  - Optimized **for time series**
  - Efficient data storage, data analysis and retrieval
  - Easy to maintain
  - Scale Horizontally
  - Easy to create complex dashboards

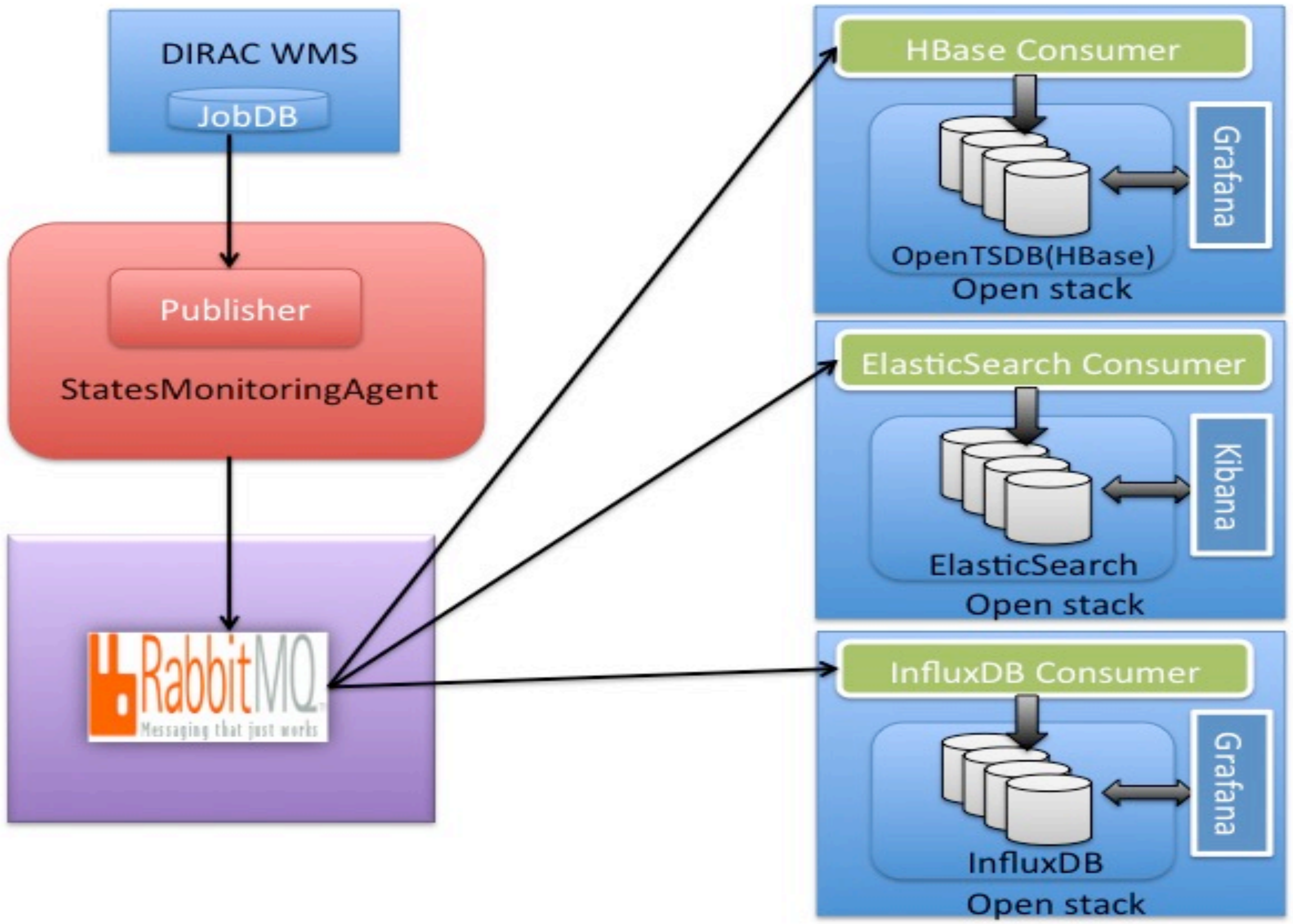


## Take advantage of new tech

- Studied storage, retrieval and analysis technologies
  - OpenTSDB
  - InfluxDB
  - Elasticsearch
- Communication:
  - Broker: RabbitMQ, ActiveMQ, …
  - Protocol: AMQP (pika) or STOMP (stomppy)
- Data visualization:
  - Grafana for InfluxDB and OpenTSDB
  - Kibana for Elasticsearch

# Test setup

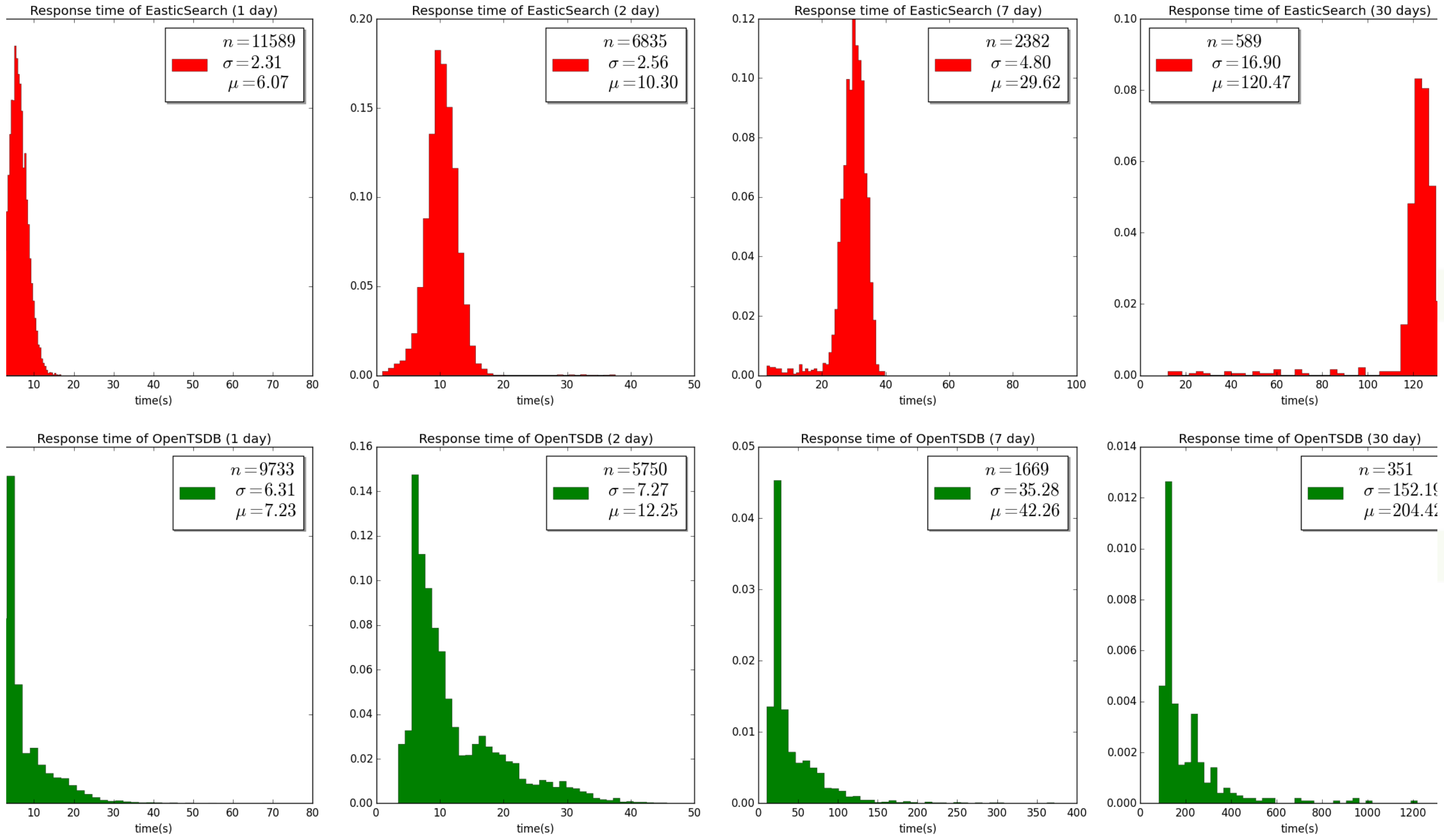
- Based on loosely coupled components



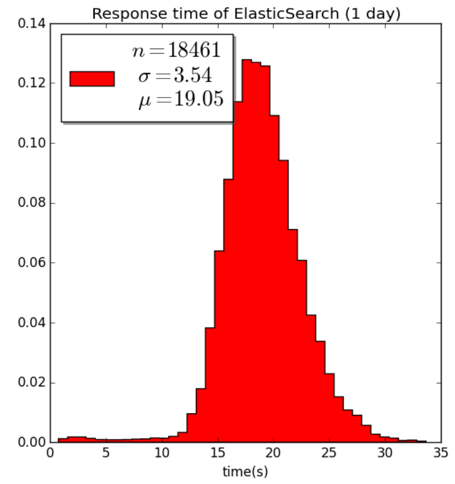
## Test setup (2)

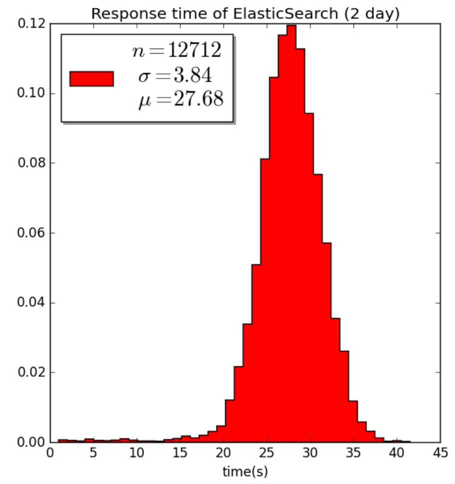
- 12 VMs provided by CERN OpenStack (3x4 nodes 4core 8GB RAM 80GB HD)
- Test conditions
  - Approximately 600 million records recorded during 1.5 month
  - 5 different queries using random query intervals
  - 10, 50 and 100 clients (python threads) are used to generate high query load
- REST APIs are used to retrieve the data from the DB
- All clients are used a random query and a random period
- All clients are continuously running parallel during 2h

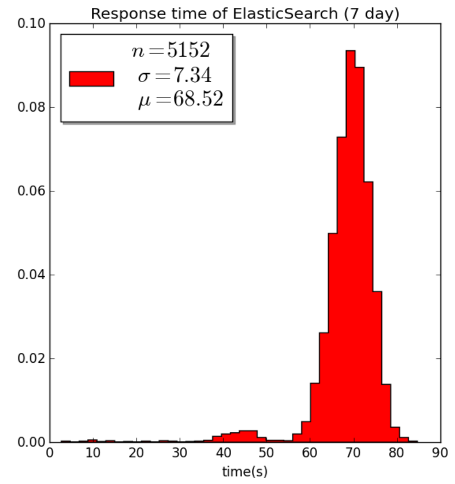
# Results with 10 clients

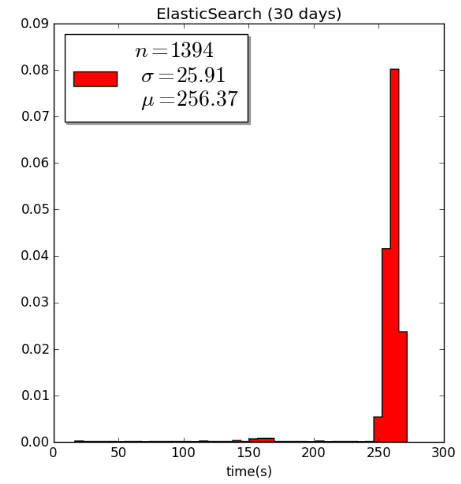


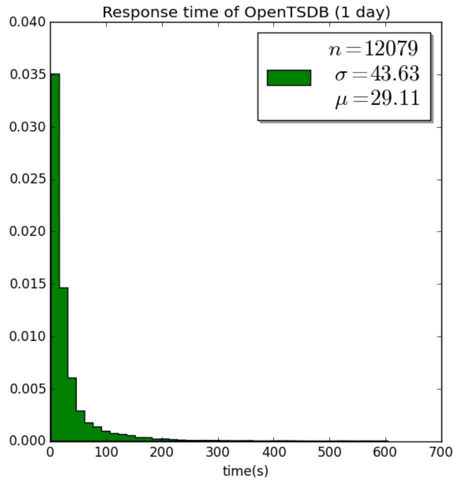
# Results with 50 clients

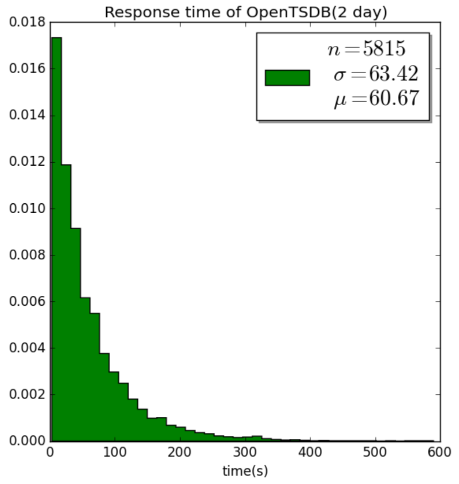


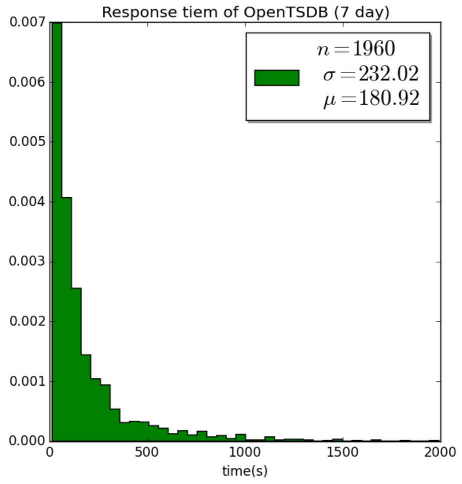


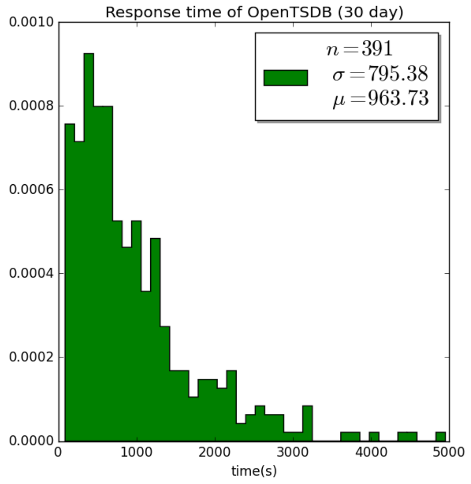




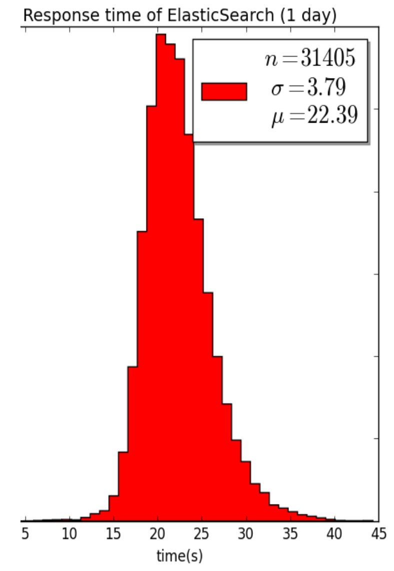


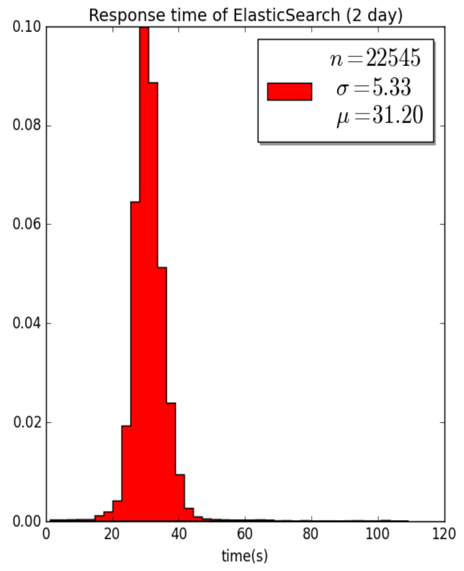


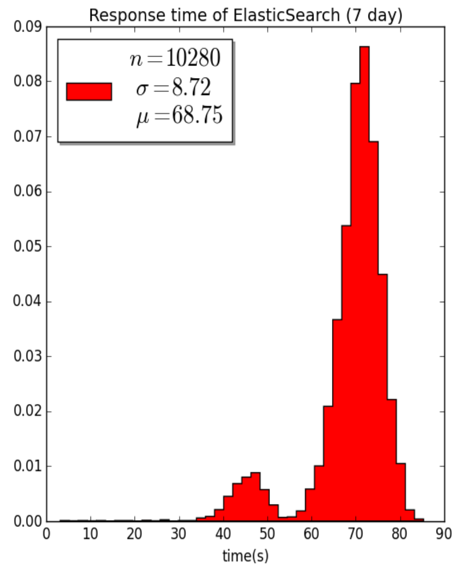


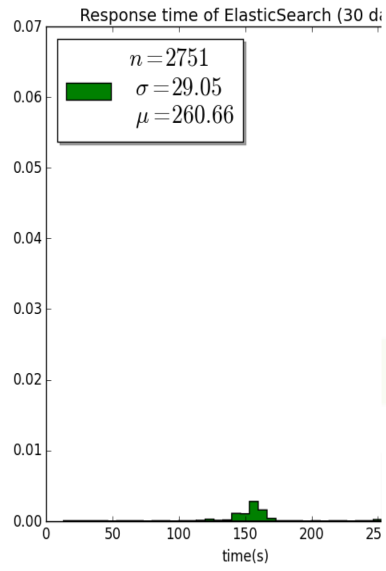


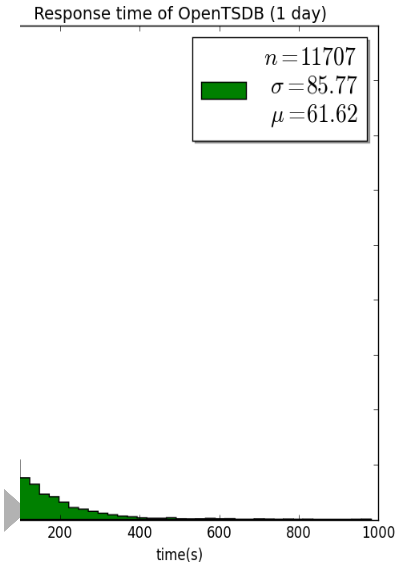
# Results with 100 clients

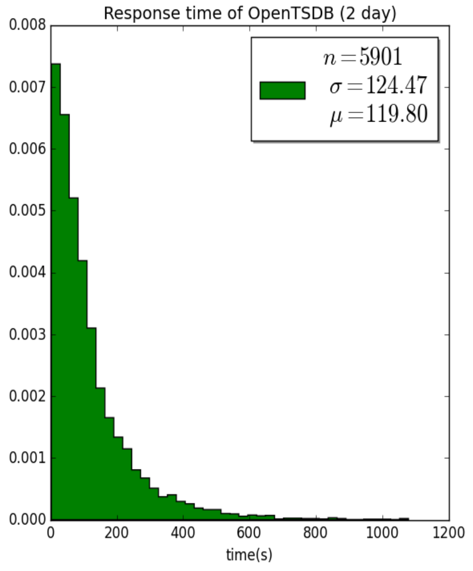


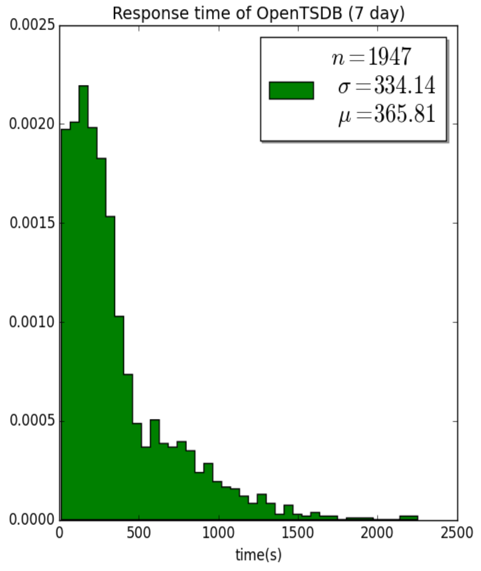


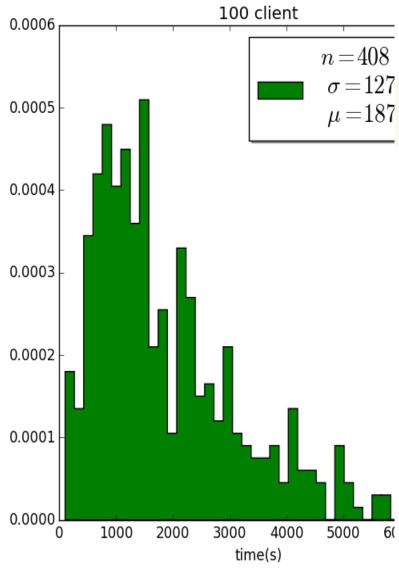




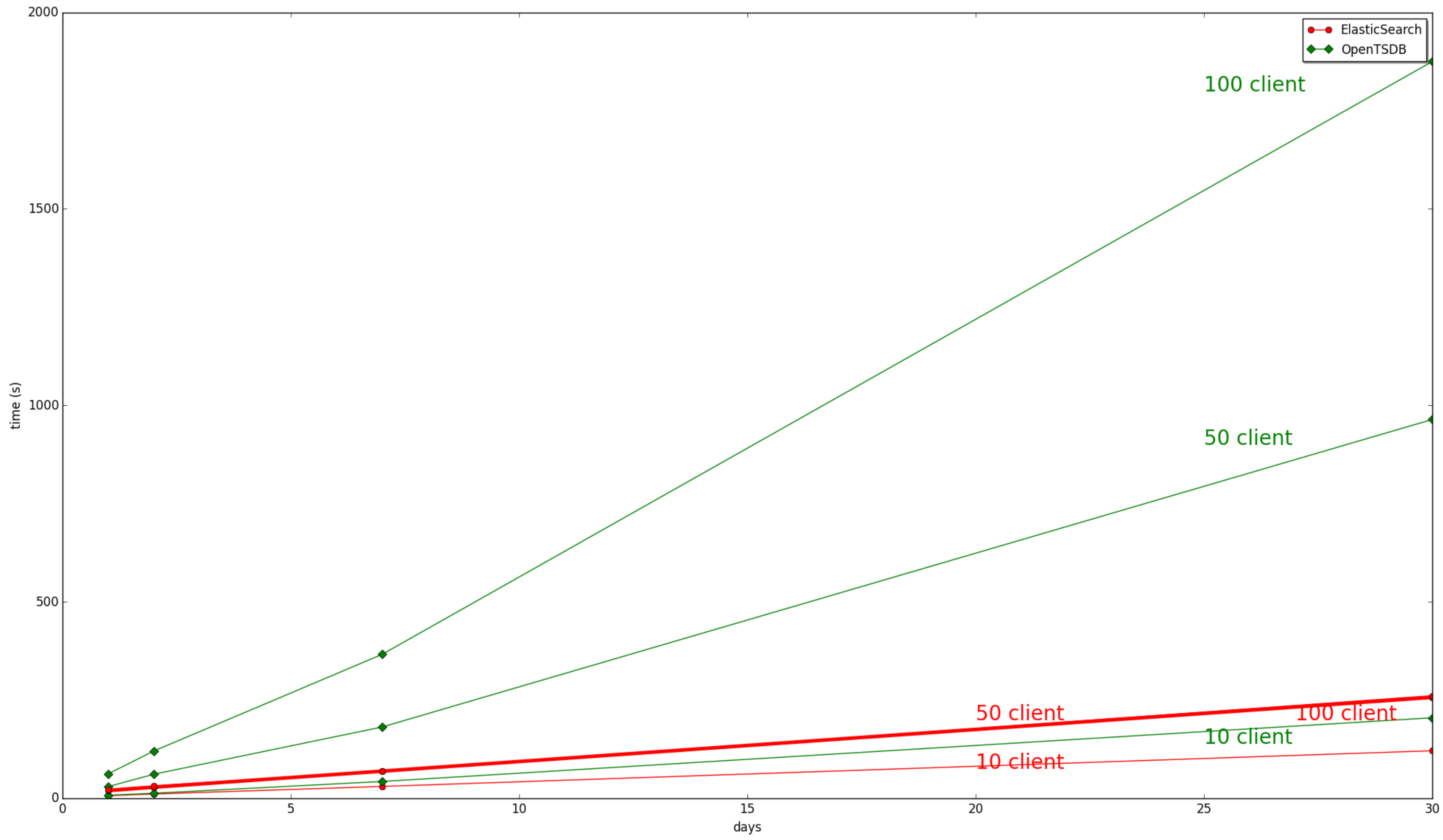






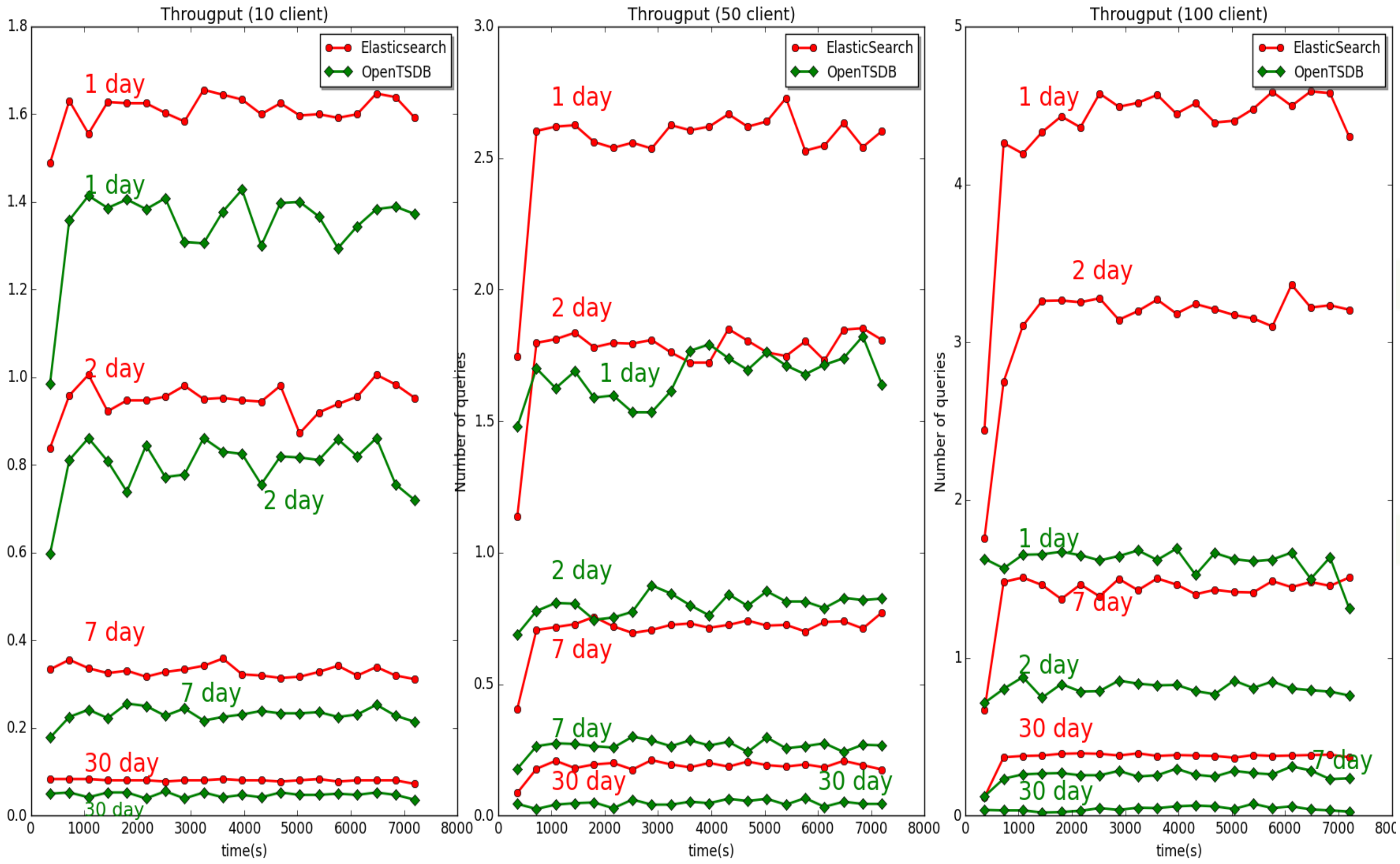


# Response time of all experiments



DIRAC THE INTERWARE

## Throughput of all experiments



## Outcome

- Settled on ElasticSearch
  - Faster than OpenTSDB and InfluxDB
  - Easy to maintain
  - Marvel is very good tool for monitoring the cluster, but it required license when the cluster is used in production (elastichq can be used instead)
  - It can be easily integrated to the DIRAC portal
  - Kibana is fulfilling our needs

## Onwards

- Authentication has to be designed and implemented
- Evaluate ActiveMQ and STOMP
- Migration from the current system
- Integrate to the current Accounting web application
- Implement a "bucketing" algorithm for old data
  - This is low priority