Future Evolution

Adrià Casajús

University of Ferrara

20150528

DIRAC
THE INTERWARE

Low disruption / Low gain

Crazy Idea

- Seems monitoring systems are popping up uncontrollably!
  - Activity System, System Adm + new monitoring component, RSS, Belle2 monitoring…
- How about we unify all the monitoring in one system?
  - Gather info, report and act based on it

Microservices
What we are doing

- Small independent processes that focus on a task organized around capabilities

- Easily replaceable and scalable

- More or less what we're doing
  - Service side OK (replaceable and scalable)
  - Agent side KO (not scalable)

Microservices

- Maybe we’re doing too many
  - ~20 DIRAC processes in one of LHCb’s WMS boxes (2CPU)
    - Most are agents

Microservices
What we are lacking

- Automation deployment and management
  - Improving
- Elasticity
  - Deploy when needed and reduce when idle
  - Fully automated
  - How can this be achieved?

Service discovery, monitoring and configuration

- Currently we’ve got self made tools for this
  - CS + Activity System + Dynamic monitoring
    - Seems new monitoring tools are popping up all around ☺
- Plenty of tools have appeared recently
  - hashicorp/consul
    - Config + discovery + service state
  - Etcd, zookeeper
    - No service state (can be done home-made)
  - Many more (nerve/synapse/smartstack, curator…)
  - No single point of failure (no master/slave)

Application Containers (Docker)

- Containers are different than VMs
  - ~0 overhead
  - Think of it as a process jail on steroids (with its own libs, dirs, config..)
- Can be moved around just like a VM
  - Don’t save state unless explicitly done by user
- Allow easy way to run apps/services/jobs
  - No dependencies

Application containers
Server side

- Ease installation of components
- Could bundle several services/agents in a container?
- Take advantage of container schedulers like
  - Kubernetes (Container scheduler)
  - Apache Mesos (Container/binary scheduler)
  - Mesosphere (OS that schedules containers)
  - CoreOS (OS that runs all apps in containers)
- Auto-scaling using container schedulers + Service discovery/config/health tools

Application containers
Client side

- RUN JOBS IN CONTAINERS!
- No user collision
- Jobs are perfectly isolated amongst themselves and vs the host
- OS/Distro independent
  - Containers have all we need to run jobs inside

Single Sign On

- We’ve got already SSO in DIRAC. You only log-in with one credential
  - We also have only one system

Multiple authentication mechanisms

- Allow users to authenticate to DIRAC via a third party auth/login service
- Many options:
  - Oauth2, Shibboleth, CAS
- Users should run under a generic proxy/certificate
  - Generic pilots anyone?
  - No glexec (or similar) since users don’t have certificates
- A user can authenticate using more than one source
  - Institute credential or google auth or cert or ….

Centralized exception reporting

- Whenever an uncatched exception appears we just log it
- Tools in the market to monitor them
  - Sentry, airbrake, exceptional, honeybadger…
- Sentry is open source and available as Saas
  - Saas free plan might be enough for us..
- As they say: Shit happens, be on top of it.
  - https://getsentry.com

Medium disruption / medium gain

Message Queues

- Async communication between producers and consumers
- No polling DBs
- Allows varying number of workers
  - Replace agents for workers/executors
- Multiple routes
  - Msgs to WMS/Optimizers, DMS/Requests, …
- Resilient and horizontally scalable

Message queues

- Many options in the market
  - RabbitMQ, ActiveMQ, Kafka, kestrel, NBQ,…
- Some of the speak the same protocol, some don't
  - RabbitMQ & ActiveMQ → AMQP and STOMP
  - Kafka, kestrel, nbq → custom
- Need to find the one that suits most to us
  - Easily maintainable
  - Replication & scaling
  - Almost zero operation time
- Check out http://queues.io/ for a mind boggling list

Agents → Consumers of events

- Instead of polling a DB an Agent should REACT to events
  - Stateless
- Spawn as many consumers as you require
  - Dynamically
- TS can react to new files
- Components can react to CS changes
- Requests based on things that happen…

Welcome to NoSQL

- Replace MySQL with a NoSQL solution
  - ☺ High availability
  - ☺ Horizontal scaling
  - ☹ No transactions
    - We don’t use many in DIRAC anyway
  - ☹ Each NoSQL node will be slower than one MySQL instance
    - But you can add more nodes…
- Already went over ElasticSearch for monitoring
  - Rebuild monitoring system around it

NoSQL

- Plenty of choices
  - Cassandra, Riak, MongoDB, Aerospike…
- Plenty of decisions to make
  - Which one suits us better?
- In any case we need to think of a replacement for MySQL
  - Oracle is only forced to maintain a GPL version until end of 2015, after that…
  - SQL alternatives: MariaDB, PostgreSQL,…

High disruption / high gain

Metadata only catalog

- Why use paths as metadata?
  - /vo/…/../prodid/taskid/…/phaseofmoon/…
- Looots of directories
  - Pain to maintain, not scalable
  - What if you want to add a need attribute?
- Metadata only catalog
  - Object store as SEs (path independent, scalable and fast)
- Get me all the files that have this prod id, with the latest task, processed yesterday…
  - Instead of list of random numbers (aka LFNs)

Graph databases

- One possible solution for metadata catalog
  - Data generates a connected graph
  - Requires investigation
- Find node that relates to this set of nodes (attributes)
- Possibilities
  - Neo4j
  - FlockDB
  - AllegroGraph
  - GraphDB
  - InfiniteGraph
  - …

Sore files in object stores

- Distributed, resilient, easy to setup
- Internally replicate data to minimize data loss
- Tipically have an AWS-like API or SWIFT-like API
- Plenty of options
  - Swift (OpenStack object store)
  - Ceph
  - XtreemFS
  - Gluster
  - MooseFS

Python

- Python 2.7 is sunsetting in 2019/2020
  - Crisis!
  - Panic!
  - Zombie apocalypse!
  - Repent! The end is nigh!

- Should we start pillaging?

Language

- Python may not be the best language for:
  - Parallel apps
    - Dreaded Global Interpreter Lock. Only one python thread at the same time
    - LHCb has +30 boxes for DIRAC
  - Distributed apps
    - All the instances require the same python version, modules and dependencies (externals anyone?)
  - Apps bigger than scripts
    - Testing python is difficult
    - Spaces vs tabs (why??)

```
myvar = 1
if random.random() > 0.1:
  print( myvar + 1 )
else:
  print( myvab + 2 )
```

Choices:

- C/C++
  - Require a ninja level knowledge to take advantage of lang
  - Not particularly designed for parallel or distributed apps

- Java/Scala
  - Same as C/C++. (java6 vs java7 vs java8…)

- Erlang
  - Perfect for parallel and distributed apps BUT
  - Not imperative. Slow learning curve for devs

Choices (2)

- Perl/Ruby/nodejs/…
  - Same faults as python
- Rust
  - Promising (designed for parallel apps)
  - But a bit complex and they just hit 1.0 (need a bit more stabilization)
- Nim (nimrod)
  - Worthy candidate (python devs will like a lot)
- Go
  - Hits the mark

Go

- Designed by google for building distributed services
- Inherent parallelism embedded in language
- Testing is embedded also into the language
- Compiles into a single static binary (easy distribution)
  - Forget about externals
- Easy to learn

Go

- Big community → Lots of stuff already there
- No coding conventions required (go fmt …)
  - No spaces/tabs problem
- Can be run on the fly
  - http://play.golang.org/
- Many companies are leaving their scripting language and migrating to Go/Erlang/Scala…
  - Dropbox used to use python, migrated to Go, got a reduction of 70% in their number of hosts