LHCb and DIRAC strategy towards the LHCb upgrade

Federico Stagni

on behalf of the LHCb distributed computing team

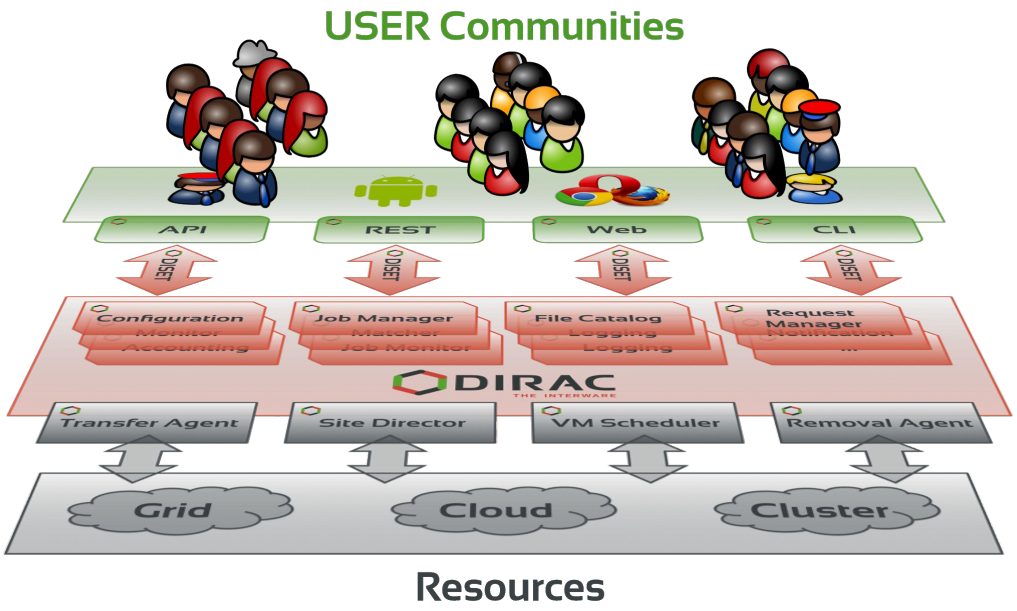
CHEP 2018, 9-13 July 2018, Sofia, Bulgaria

Disclaimer

- This is a strategy talk
- All technicalities are left in backup slides
  - happy to answer questions later

DIRAC: the interware

- A software framework for distributed computing
- A **complete** solution to one (or more) user community
- Builds a layer between users and resources



- Started as an LHCb project, experiment-agnostic in 2009
- Developed by communities, for communities
  - Open source (GPL3+), GitHub hosted, python 2.7
  - No dedicated funding for the development of the “Vanilla” project
  - Publicly documented, active assistance forum, yearly users workshops, open developers meetings
  - 4 FTE as core developers, a dozen contributing developers
- The DIRAC consortium as representing body

Users/communities/VOs

A *framework* shared by multiple experiments/projects, both inside HEP, astronomy, and life science

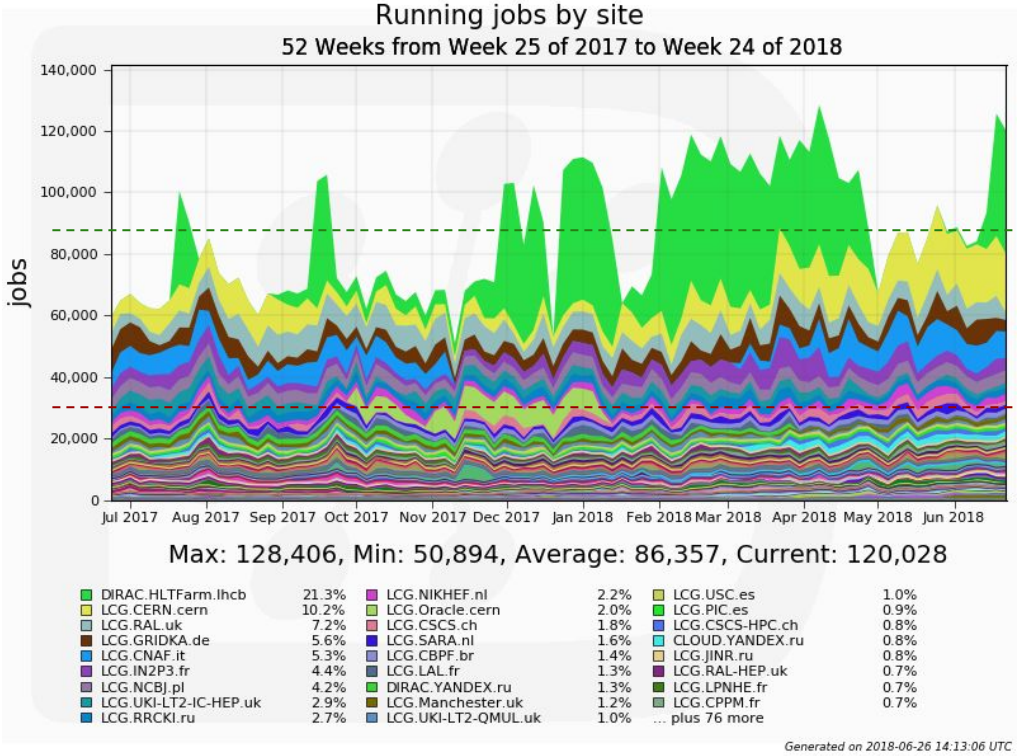
<u>Experiment agnostic</u>
<u>Extensible</u>
<u>Flexible</u>

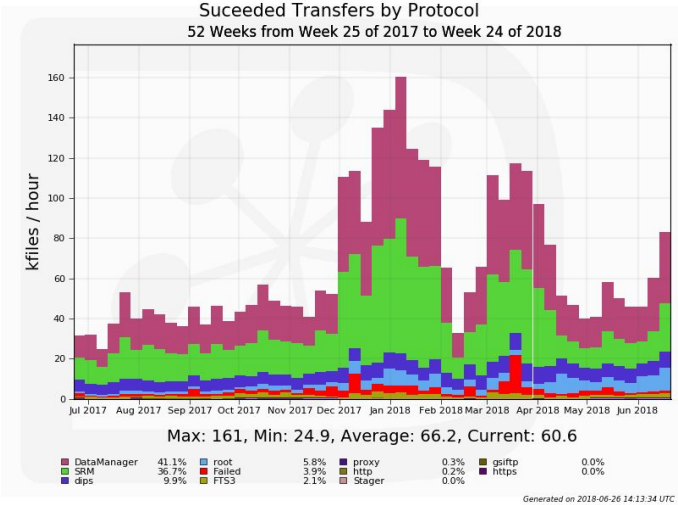
LHCb and DIRAC

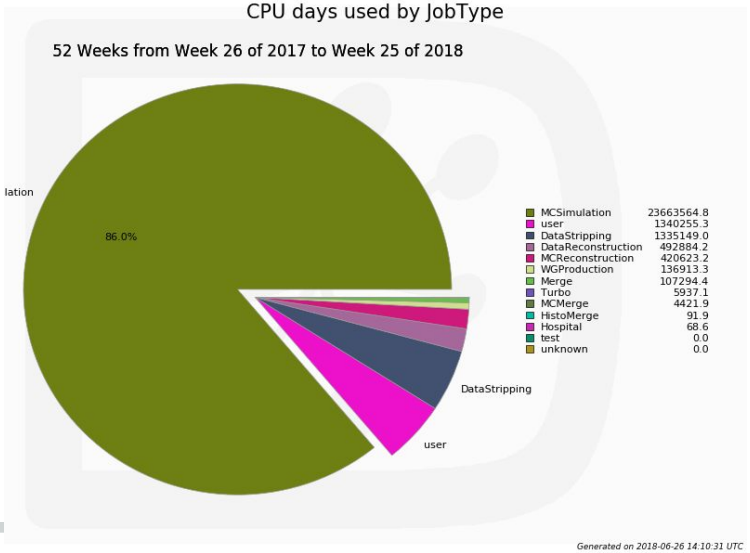
- LHCb uses DIRAC for managing all distributed computing activities
  - Workload, Data, Productions, etc...
- LHCb analysts use Ganga (which interfaces to DIRAC) or DIRAC APIs directly

LHCb is the experiment that stresses DIRAC functionalities the most

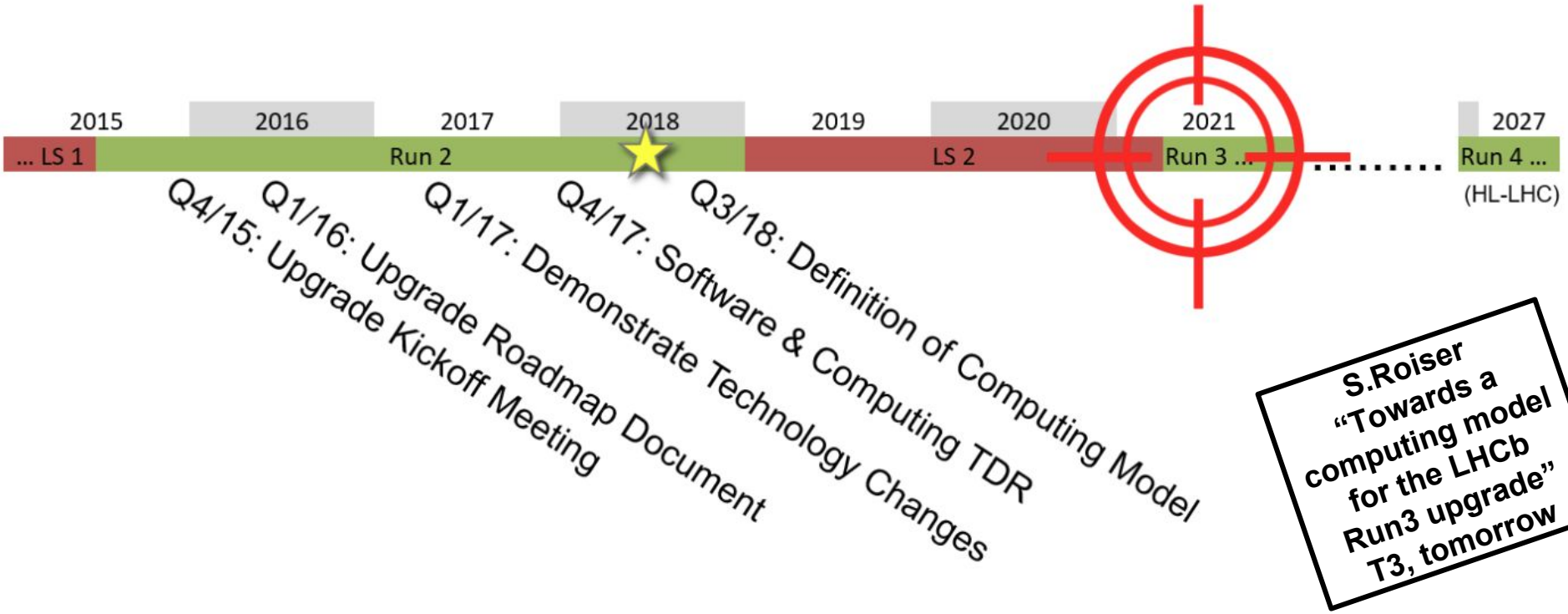
LHCb DIRAC Activities overview







The LHCb upgrade



S.Roiser "Towards a computing model for the LHCb Run3 upgrade" T3, tomorrow

- **Computing model being defined**
- **DIRAC needs to be able to implement it, and sustain an increased load**
- **Resources "crisis" → grab what you can!**

Exploiting computing resources

Pilots are the "federators"

Send them

as "pilot jobs" (via a CE)

Or just **Run them!**

e.g. as part of the contextualization of a (V)M

A.McNab "The LHCb DIRAC Containers" T7, tomorrow

few "generations" of pilots

Running the service: LHCb DIRAC Pillars

- With DIRAC, LHCb operates a service
  - need to keep a running system working, with *continuity*
- We don't see the need for a revolution
  - The system will keep evolving gradually
    - in a backward compatible way
  - Introducing new/better/faster stuff
    - Users should not notice about (most of) them

- Usability for the users
- Scalability for the services is necessary

Scalability is about

1. Traffic growth

   how many messages

   → DIRAC architecture and framework

2. Dataset growth

   how much data

   → RDBMS, NoSQL...

3. Maintainability

   system and code

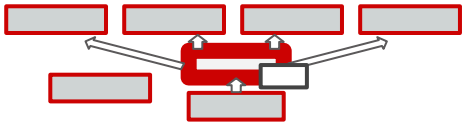
   → Software engineering practices

Traffic growth:
DIRAC architecture

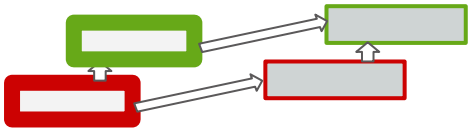
we're good

more or less

DIRAC is a (micro)service architecture



Horizontal and vertical extensibility



...so yeah, no revolutions ahead

...still… things to do…

Traffic growth:
DIRAC Framework

The DIRAC Core and Framework has been developed +10 years ago:

- logging
- DISET
  - sockets + SSL + DEncode
- ...

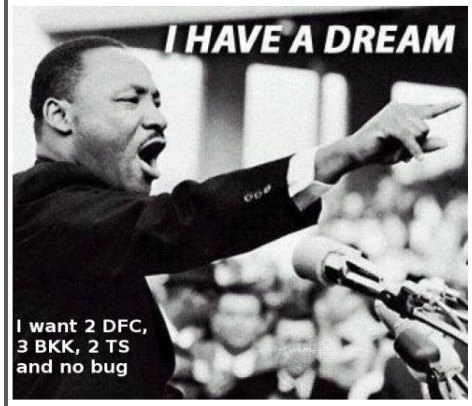
and now lots of stuff in there is available/maintained elsewhere

- is it worth/better than what we have now?
  - ***partly, yes!***
  - So, we are moving/adapting
    - all under-the-hood
      - ask for details
- →reducing the codebase

Traffic growth:
enter the orchestrators

- **HW** scalability: more (micro)services
  - Supposing a good load balancer, this is not necessarily bad
    - self-tuning system?
  - Enter in the beautiful world of the ***orchestrators***...

I have a dream



Let “something” run it “somewhere” for you

DIRAC on Mesos/Kubernetes?
<u>Almost</u>, but...

Traffic growth:
Message Queues

Important topic, for which there's a poster!

#215 by W. Krzemien

Dataset growth: Databases

- RDBMS:
  - MySQL OK
  - Oracle (Bookkeeping) OK
- NoSQL:
  - ElasticSearch (in production)
    - Other options tested, converged to ES
- Object and block storages

Maintainability:
Python 3

- … yes, one day
  - not (really) tomorrow, but we started
- we have been polishing the code for long time now
  - so, 2to3 (modernize) won't explode
- wide, deep, testing is fundamental
  - a testing and certification process is in place

→ but we need our (several) dependencies to move first!

User analysis

- Ganga will be kept
- Centralized productions for WG analysis

Summary

- LHCb uses DIRAC for all its distributed computing activities, and will keep using it for Run3 and beyond
  - DIRAC users and developers from other communities, LHCb the one that stress its capabilities the most
- Focus on usability, flexibility, scalability
  - Already flexible
  - Scalability: traffic and dataset growth, and maintainability
- No need for a revolution → constant evolution
  - strategy defined few years ago
  - several developments started, some completed, others we'd like to start
  - what LHCb is doing and will for DIRAC will benefit all other communities too.

Questions/comments

?

- Web: http://diracgrid.org
- Docs: http://dirac.readthedocs.io
- Forum: https://groups.google.com/forum/#!forum/diracgrid-forum
- Code, wiki, and issues (DIRAC): https://github.com/DIRACGrid/
- Code (LHCb DIRAC): https://gitlab.cern.ch/lhcb-dirac/
- 8th DIRAC Users Workshop: https://indico.cern.ch/event/676817/
- Devs' meetings: https://indico.cern.ch/category/4205/

BACKUPS

Reminders

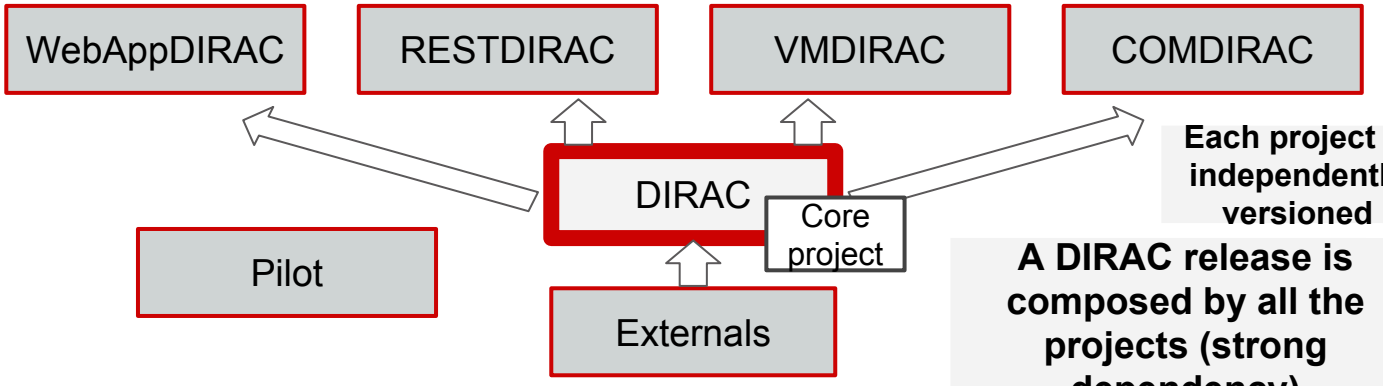
- DIRAC is a fully open source project
  - Extended in LHCb
- Used by 40+ communities
- 220K+130K lines of python 2
    - Plus some .sh, .js

Experiment agnostic, and extensibility

"Horizontal" extensibility

-

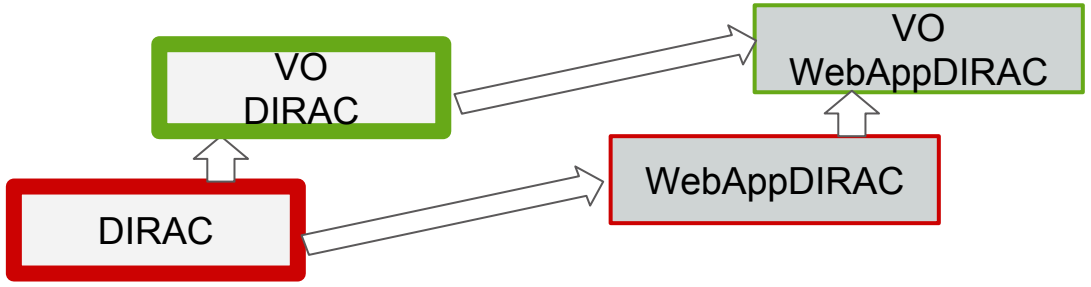
For specific requirements



"Vertical" extensibility

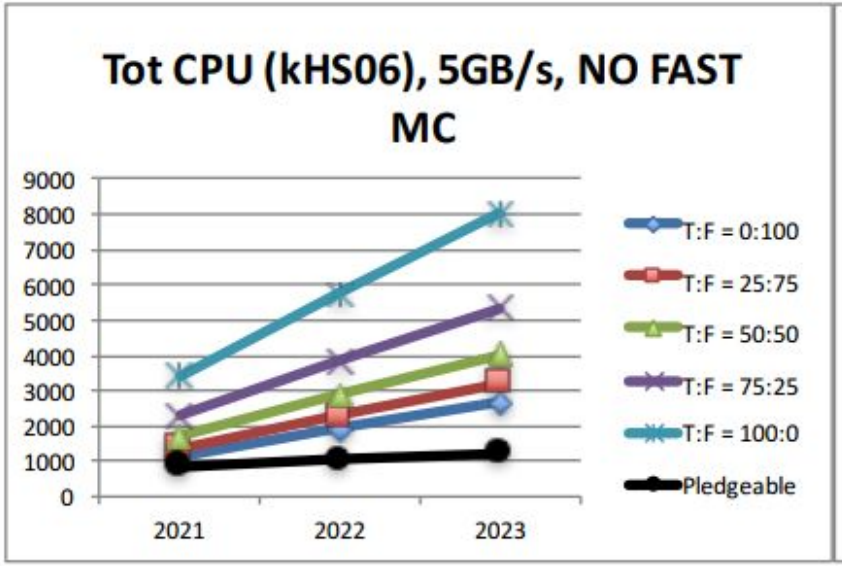
-

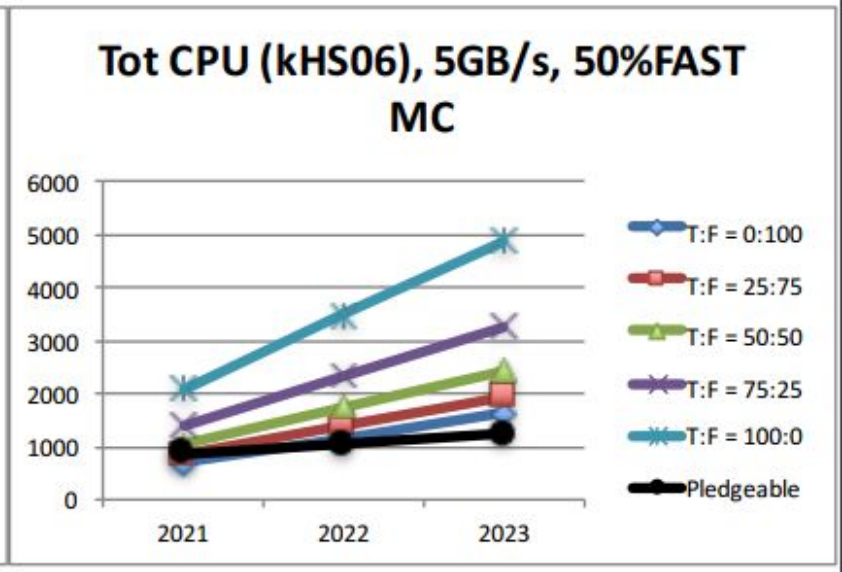
Community driven

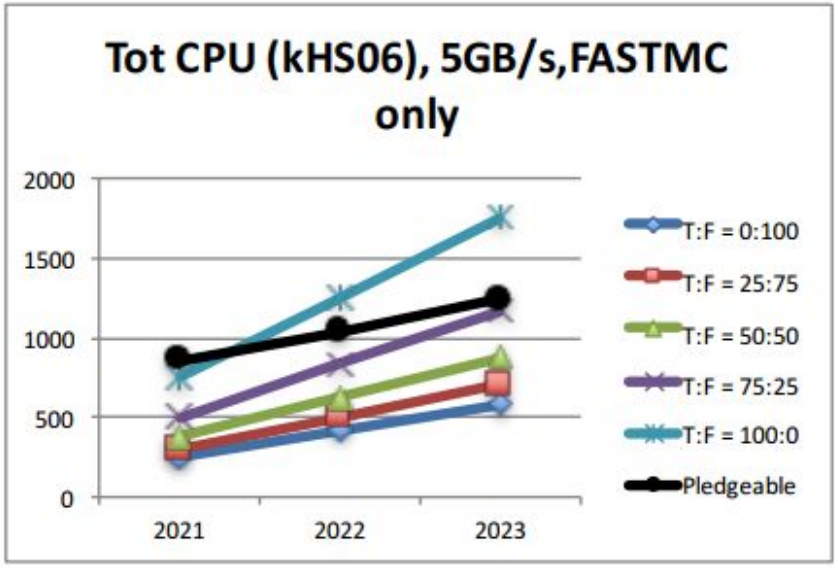


Concezio

Offline CPU for simulation @ 100% of real data







- Resources for data processing between 100 and 200 kHS06 (not shown)
- Only full simulation: factors off the pledgeable resources
- Only fast simulation: resources within the pledge envelope – some full MC still feasible

Resources

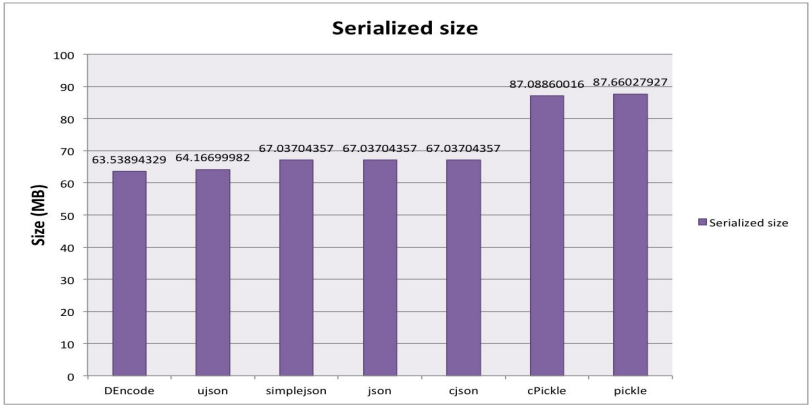
How many resources will we need?

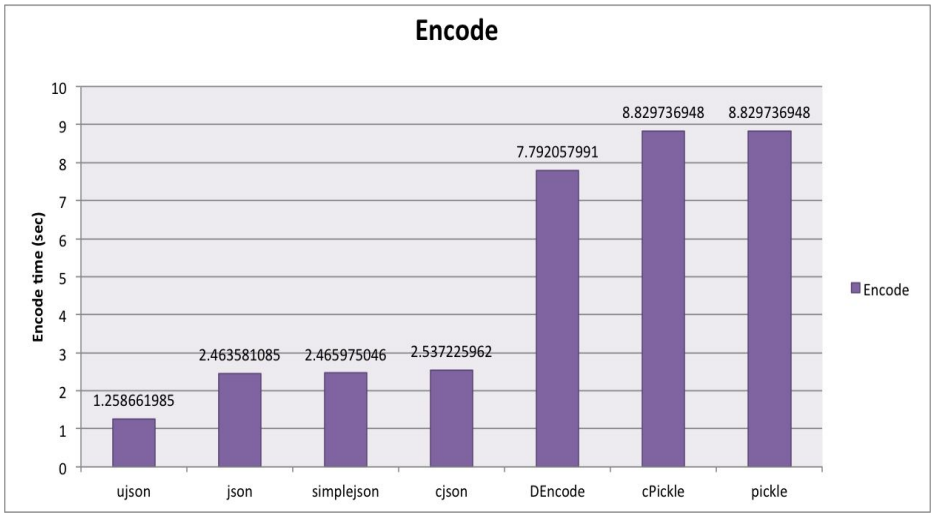
- Playing with numbers
  - hard to predict, factors in differences.
- But, however we put it, LHCb in Run3 will require way more computing, disk, and tape resources than what we have now.
  - the approach from funding agencies need to change
- Non-pledged, opportunistic resources help, but won't save us
  - and anyway they are only computing resources

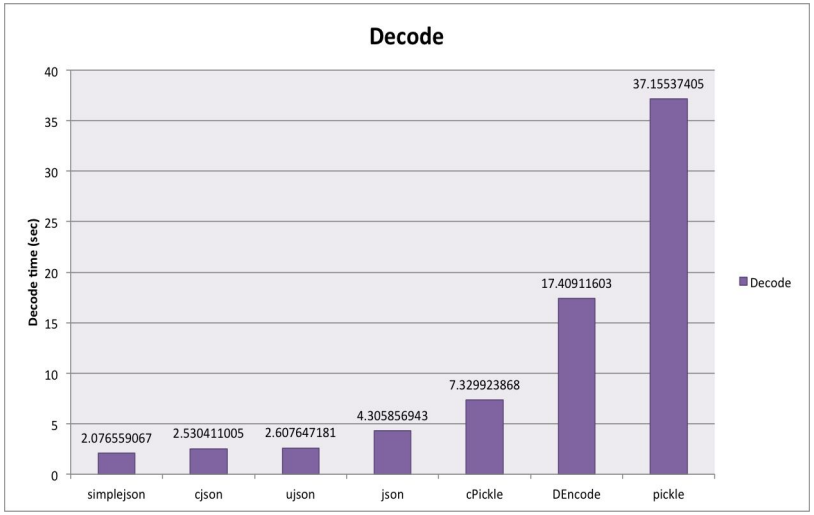
Encoding/decoding

What if we change DEncode?

Test: 178k files and their metadata







DIRAC pilots

1. Install a DIRAC client
   - together with dependencies
     - the “container” is shipped → a “container” is not necessarily an image
2. Self-discover WN capabilities
   - Including CPU power and capabilities
     - Using DB12 or MJF
     - And #processors
   - And memory
3. Use a “JobAgent” to match the capabilities of the WN with the requirements of the waiting jobs.
4. Send monitoring info
   - A list of messages like
     - "I've booted up" …
     - "I found the DIRAC pilot ok" ...
     - "I'm about to shutdown"...
   - Self-upload their own logs before shutting down

Traffic growth:
DIRAC Core and Framework

The DIRAC Core and Framework has been developed ~10 years ago, and now lots of stuff in the DIRAC framework is available/maintained elsewhere

- this is already technology
  - is it worth/better than what we have now?
    - ***partly, yes!***
  - gLogger → python logging [DONE]
    - and plugins on the shelf!
  - pyGSI → M2Crypto [IN PROGRESS… STOPPED?]
  - dips → https [STARTED]
    - see later

DIPS → HTTPS

- Migrate from DISET to HTTPS
  - Ready for python 3
  - Standard way to call a server
  - Easy to understand
  - Big community behind

- Progressively remove DISET
  - **Transparency** & **Backward and forward compatibility**
  - Keep DIRAC internal functions (Monitoring, proxy, authentication/authorization, …)
  - Prepare work for complete integration with Tornado

More info

Requirements on Traffic

Assumption:

increase of one order of magnitude

- Services: **~OK** if lots more services
  - → that "~" is meaningful!
- Agents: **KO**

Limitations of agents

- Polling&Pulling
- Not real time executors

For today's implementation:

- Parallelization is hard
- Multiple instances may not be possible
  - Lots of work anyway

→ NOT scalable

Scalability in mind

- **HW** scalability: more (micro)services
  - Supposing a good load balancer, this is not necessarily bad
    - self-tuning system?
- **SW** scalability:
  - probably need to change/replace (big?) part of DISET

Message Queues

- MQs are in DIRAC
  - For failover purposes
  - Consumers as DIRAC components → RFC
- Push, not pull
- We can replace several agents with Consumers
  - and also (especially?) executors
  - Agents, executors, consumers as a single component?
  - … what about trying with this guy?
    - http://python-rq.org/
    - a nice project...

RQ: Redis Queue

Dataset growth:
Object, block storage

Advocated as “scalable solutions”

Use cases:

- Logs
- SandBox

which are static and unstructured data

Need a DIRAC SE on top of an object storage, e.g., CEPH, or OpenStack Swift

...but again, these are mostly Dev-Ops issues.

Timeline

partial!

	16 Q4	17 Q1	17 Q2	17 Q3	17 Q4	18 Q1	18 Q2	18 Q3	18 Q4	19 Q1	19 Q2	19 Q3	19 Q4	20 Q1	20 Q2	20 Q3	20 Q4
No-SQL integrat.	X							X									
MQs integrat.	X								X								
New logging					X												
Python 3 plans									X					X			
34 High availab.																	X