Common Analysis Framework for ATLAS and CMS

Feasibility study

Fernando H. Barreiro Megino
Mattia Cinquilli
Daniele Spiga
Daniel C. van der Ster
CERN IT-ES-VOS

What is it?

- Initiative from **CERN IT-ES, ATLAS** and **CMS** for a common analysis framework started March 13 2012
- Assess the potential for using common components for distributed analysis, based on elements from **PanDA** and **glideInWMS**
- **Initial plan**
  1. **Feasibility study** - Mandate: http://cern.ch/go/9mNC
     - Analyze architectures of both experiment's analysis frameworks
     - Identify interfaces to external systems
     - Identify what can be reused
     - How much effort is it?
     - Identify show-stoppers
  2. **Functionality study**
     - What do ATLAS and CMS gain and loose in terms of functionality by adopting a common framework
  3. **Operations study**
     - What is the impact on the cost of operating various proposals
- A common analysis framework could lead the way to further commonalities and collaboration between the experiments in the future

Strategy for the Feasibility Study

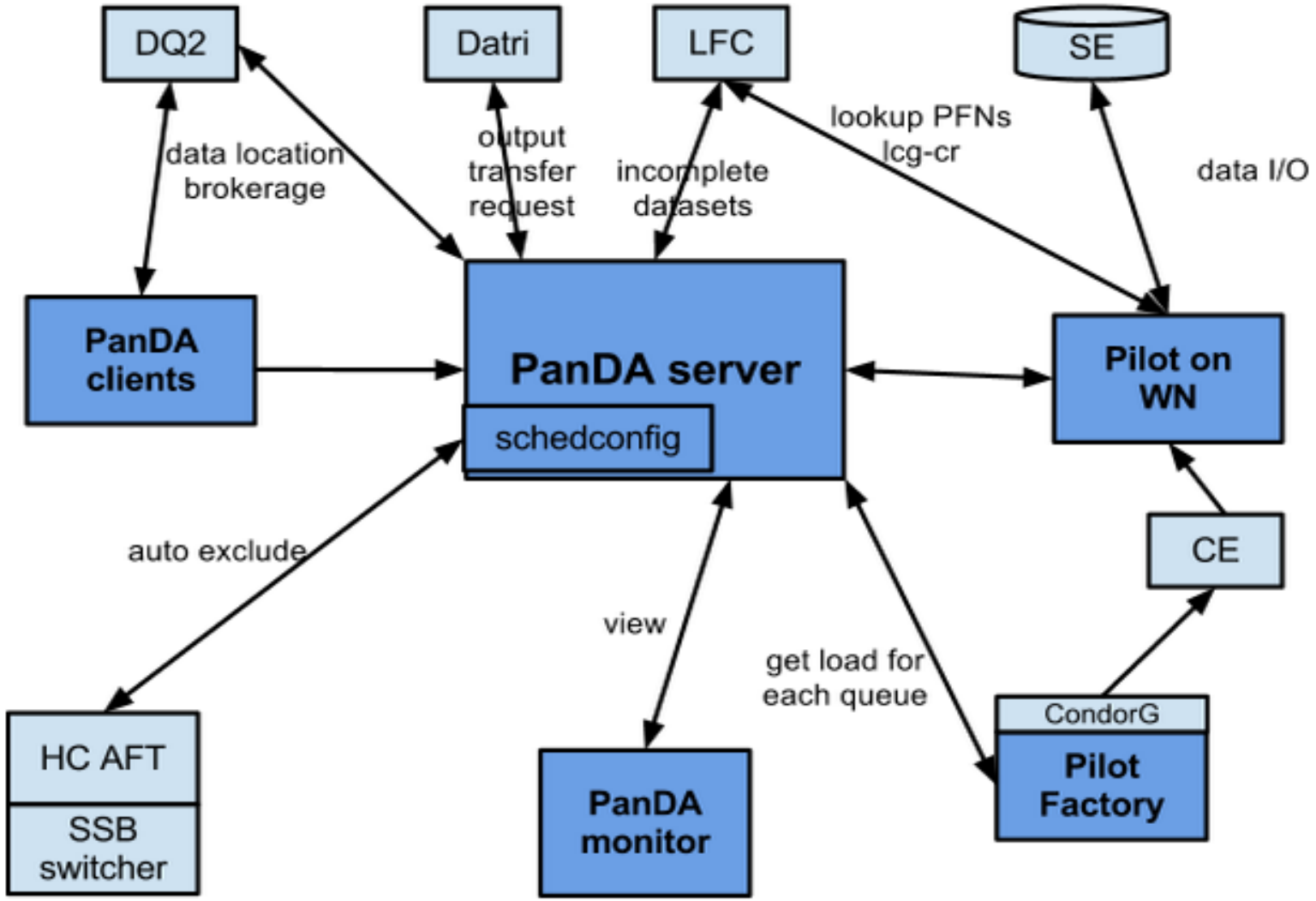
- Carried out by small working group from CERN IT-ES
- Organize meetings with experts to discuss subcomponents
- Track discussions in http://cern.ch/go/8Z8Q
  - Possibility for everybody to contribute and raise questions asynchronously
- Report on a ~weekly basis for open discussion
- **Final delivery date:** Document expected before CHEP 2012

What have we done so far?

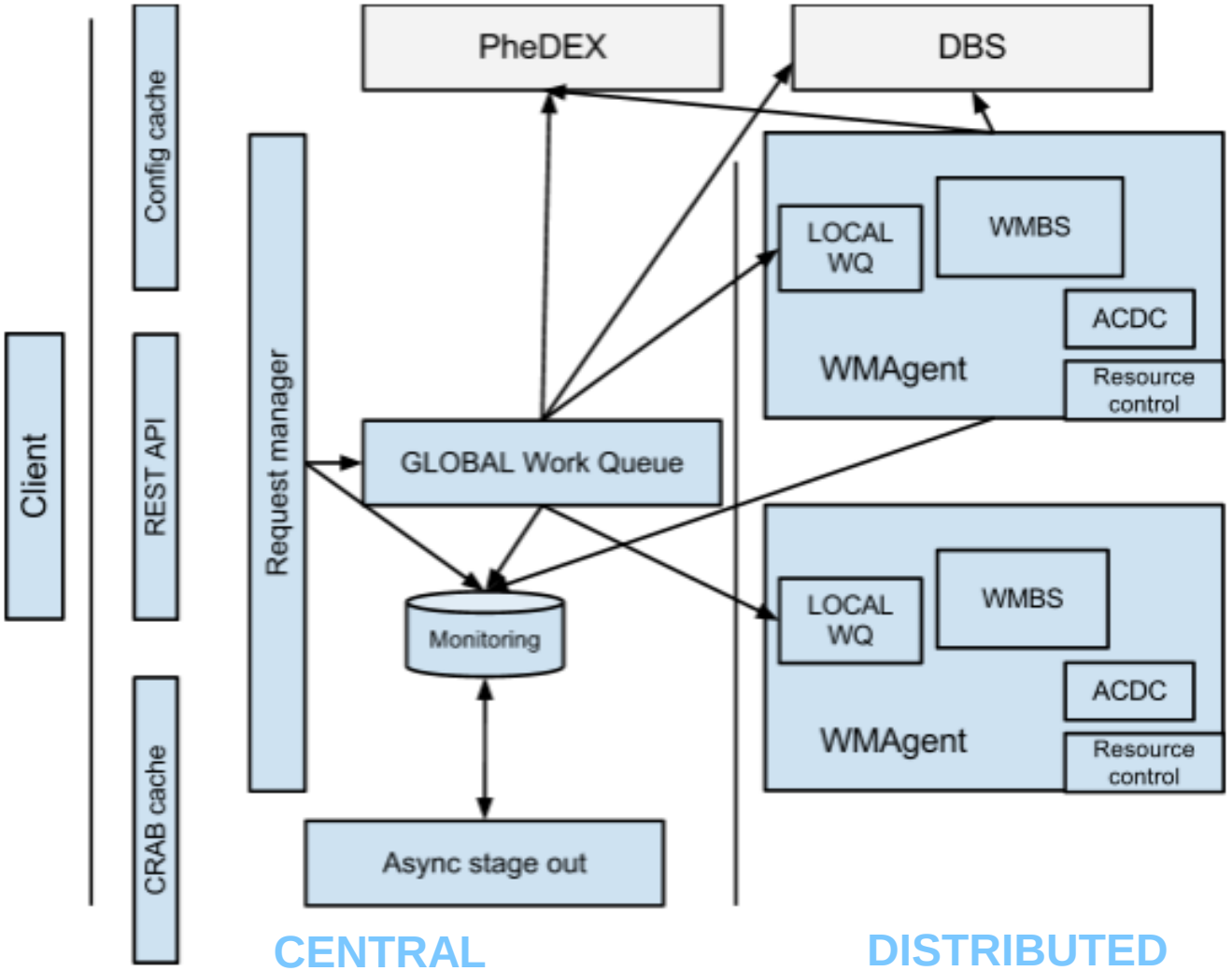
1. Compare ATLAS & CMS analysis workflows and identify main components
2. Further studies with experts
   i. Server side
   ii. Pilot factories
   iii. Pilots
   iv. GlideInWMS
3. Wrote summaries and had follow-up meetings

Overview of the the ATLAS and CMS analysis frameworks

PanDA architecture



CMS analysis framework



Resource management and brokerage

	ATLAS	CMS
Site occupancy	PanDA job table (**global view**)	Tracked by WMAgent (**local view**)

Brokerage	• Client discovers data locations
	• Followed by load based site brokering based on weight function
	  • Site capacity measured dynamically
	• **PanDA picks best site at submit time**
	• **PanDA tries to process whole dataset at one site**
		• GlobalWQ asks PheDEX/DBS data locations
		• Either
		  • WMAgent assigns based on static, local pledges
		  • Delegate to WMSes to decide the final site
		• **CMS sends a list of sites to WMS**
		• **CMS will spread across sites**

PD2P and rebrokerage

	ATLAS	CMS
Dynamic Data Placement	• ATLAS has a data distribution/pre-placement model which relies on dynamic data placement • PD2P: When a jobset is submitted, PanDA can decide to trigger a replica request	-
Rebrokerage	Jobs waiting longer than *x* hours can be reassigned to another site	• Locations for jobsets in GlobalWQ are continuously refreshed • Once the job is in the LocalWQ locations are fixed

Priorities and fairshares

	ATLAS	CMS
Priorities and fairshares	• Users get *x* CPU hours per 24h • Additional jobs are de-prioritized • Priority boosts/beyond pledge for users and groups at particular resources • @ submission: Jobs in a jobset get decreasing priorities (so that a few run right away to check for errors) • Waiting jobs: Job priority increases while jobs wait to prevent starvation • Retried jobs get lower priority to delay slightly • Prod/analy balance set at site level	• Priority is set by operators • RequestManager processes requests in order of priority • GlobalWQ fetches in order of priority • Global and Local WQs are FIFO • Prod/analy balance set at site level

Data handling in the server

	ATLAS	CMS
Input	• Pilot queries LFC to get PFNs • Flexible input data handling configured in schedconfig • Copy2scratch vs streaming I/O	• Input handling completely delegated to CMSSW • CMSSW uses Trivial File Catalogue
Output	• DQ2 for detector, simulated and user data • Copied to local SE by Pilot • Registered by the client • Optional additional copies via DaTRI	• DBS/PheDEX primarily for detector data • CRAB handles asynchronous stage out and optional DBS publication

Bookkeeping and redundancy

	ATLAS	CMS
Bookkeeping	• CLI for job/task bookkeeping and WWW PanDA monitor/Dashboard historical jobs • CLI to kill and retry jobsets • Jobset progress tracked in PandaDB (i.e. which files have been read)	• Client to kill and retry request • WMAgent handles retrial of jobs based on ACDC (i.e. which files are left to process)

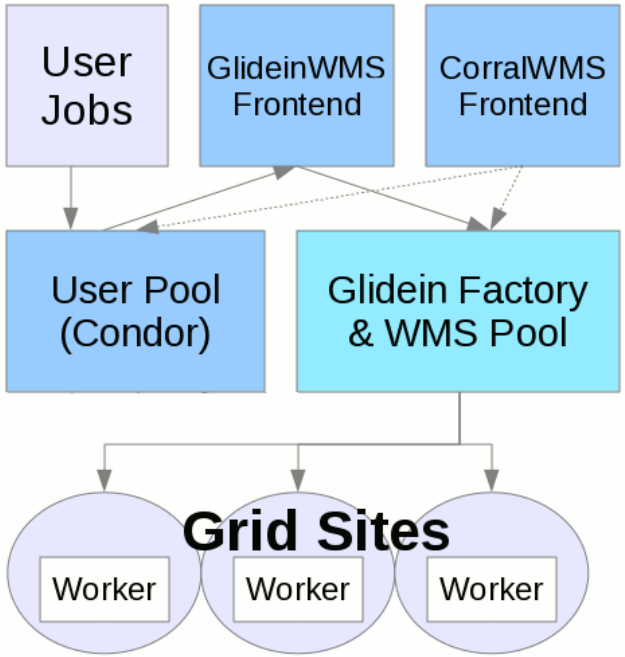
Redundancy	• PanDA@CERN is single point of failure
	• CERN Outage:
	  • No new jobs
	  • Running jobs ~OK
	  • Completing jobs may fail
		• Distributed with *n* independent queues with enough work for one day
		• CERN Outage:
		  • No new jobs

PanDA pilot, AutoPyFactory and GlideInWMS

PanDA Pilot and AutoPyFactory

- PanDA Pilot
  - Rather ATLAS specific
  - A next generation pilot could put VO specifics into modules
    - Environment Setup Module
    - Data movers (mainly for outputs)
  - glExec user switching: implemented but not used
    - Not known if MyProxy can handle queries from all WNs
    - US sites currently require pilot credential to write outputs
  - PoC could run the CRAB wrapper in the pilot (*prun* job)
- AutoPyFactory (v2)
  - Multi-threaded (one thread per queue)
  - Modular and ATLAS agnostic. Plugin approach:
    - WMSStatus Plugin: e.g. what is the state of the WMS (Panda)
    - Sched Plugin: calculate how many pilots to submit
    - BatchStatus and Submit plugins: submit and monitor pilots (Condor-G, Condor local…)
  - Flexible proxy management (per-queue proxies)
  - **The *interesting* part will be combining APF with GlideInWMS…**

GlideInWMS Overview



- Build a distributed Condor pool which looks like a local batch system
- GlideInWMS automates submission of Condor GlideIns according to user jobs

- Users (VOs) submit to a local Condor *schedd*; a frontend polls the user *schedd* and tells a GlideIn Factory to send GlideIns via CondorG to the grid.
  - GlideIns run a condor *startd* on the WN which connects back to the user pool

- Features:
  - Credential management handled by Condor
  - glExec id switching
  - Condor scheduling and fairshare between users and groups
  - Whole node scheduling
  - SSH-to-job
  - *Preemption*

Animation taken from
http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/doc.prd/index.html

GlideInWMS and CMS

- CMS is using GlideInWMS with CRAB 2 and testing with v3
  - Each CRAB 2 server / WMAgent has a local *schedd*
- CRAB server / WMAgent injects jobs (with full payload) to the *schedd*
- Using simple condor matchmaking: jobs run in FIFO order

- Condor itself has some scaling limits (provided by Igor, not definitive)

Component	Limiting factor	Observed limit
Schedd	Memory	60k jobs on 64GB node
Collector	Memory	90k jobs on 24GB node
Negotiator	CPU	40k jobs, depending on complication of matchmaking expressions

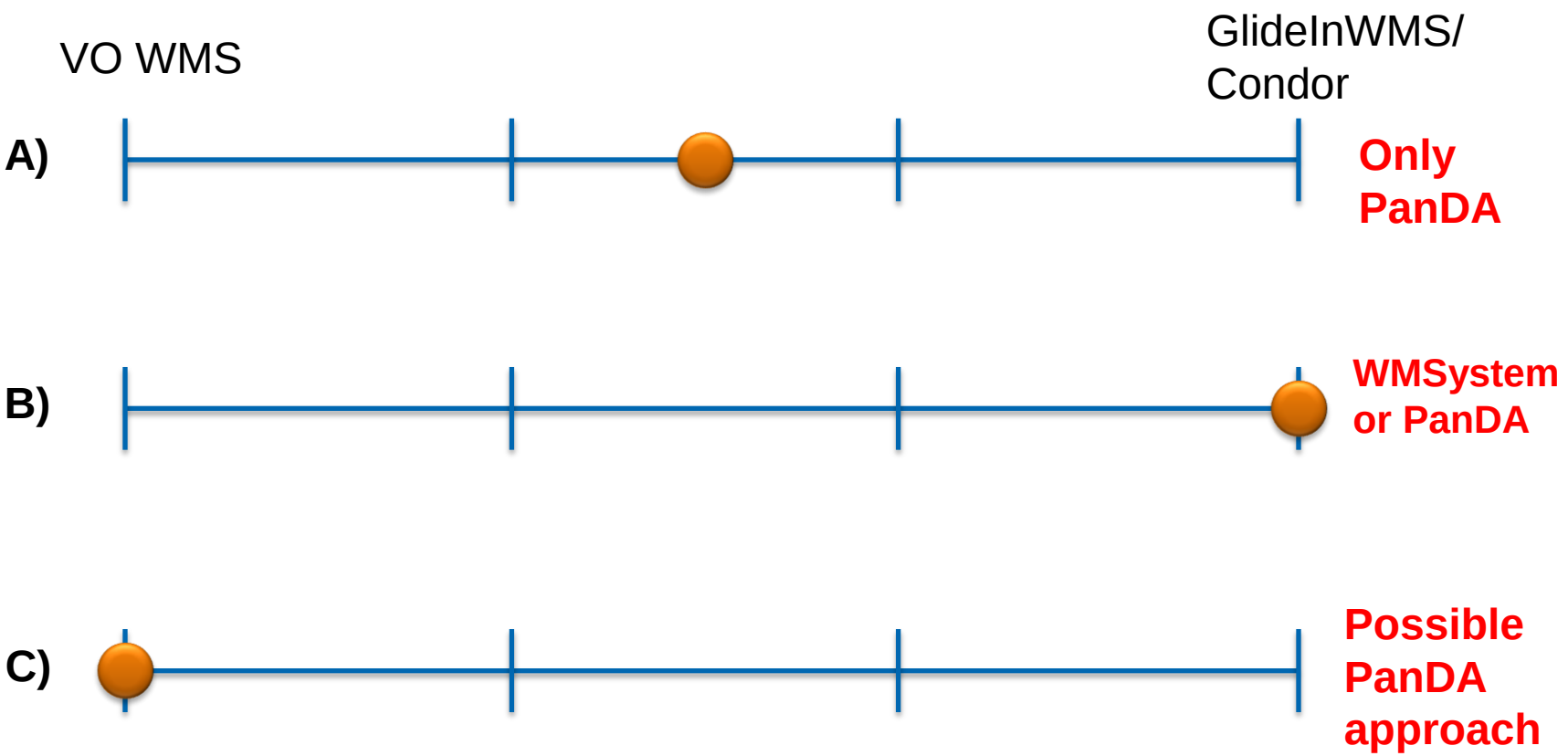
- CMS architecture allows to replicate the CRAB server / WM agent to scale up:
  - Currently ~7 agents, limit of ~20k per schedd

GlideInWMS and PanDA

- ATLAS is testing the integration of PanDA and GlideInWMS *(see Rod's presentation)*

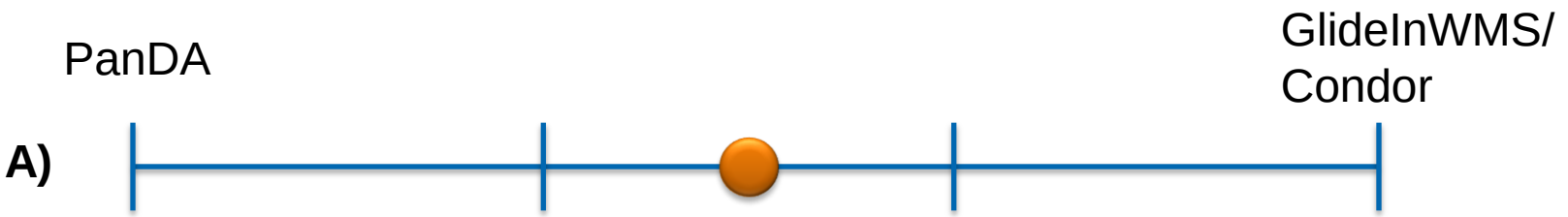
- *Schedd* is ran on the current pilot factory machines
- Submit pilots per site-user pair
  - Check activated jobs in the panda queues
  - Retrieve user proxy from MyProxy (+ cache it)
  - The job submitted to Condor will run only one user's jobs
    - Panda server will give it the user's job with the highest priority
- VO-frontend watches the pilot factory *schedd*'s
  - UCSD submit glideins to run the queued jobs

Integrating with GlideInWMS Different Scenarios



Scale shows which service is scheduling the jobs

Scenario A: Rod's approach



- AutoPyFactory (one thread per site/user) submitting to a VO Condor *schedd* (with user's credential).
- GlideIn factory submits appropriate glideins to the grid

- Implications: Two knobs to control the job schedule
  - **Condor** handles **inter-user priority**: fair share between users/groups
  - **PanDA** handles **intra-user priority**: which of a user's jobs to run next (based on priority)
  - Current Panda share logic/functionality would need to be converted to Condor
  - APF needs development to watch site/user queues

Scenario B: CMS Approach

- This is a *natural* usage of GlideInWMS
- Thin layer between WMS and GlideInWMS: a “job factory” submits jobs with the payload attached (e.g. crab wrapper.sh) , or alternatively with generic payload but specific job ID attached (e.g. panda pilot_wrapper.sh –pandaid 1234)

- Result:
  - Almost all of the job scheduling is handled by Condor (but jobs could still be submitted with priorities)
  - If applied to ATLAS, job rebrokerage would become more complicated: Need “job-killer” that knows mapping between PanDA and Condor job ids

Scenario C

- ATLAS is still investigating AutoPyFactory as it is today with **pilot+glExec** to solve the security issue.
  - MyProxy scalability is being investigated
    - Local client/server limit ~10 Hz (~1M/day)
    - Multiple clients separated from server easily >25Hz (John Hover)

- AutoPyFactory and GlideInFactory have significant conceptual overlaps

- *Would lose out on Condor features*
- *Not a common approach*

Torre's proposal: Pool of credentials

- **Use a pool of credentials**
  - As large as number of users could run on a site simultaneously
  - Assign a particular role to these credentials
  - Implement framework to assign credentials to users
  - Site never sees the real user, but there is increased traceability
  - Removes dependency on MyProxy
- All benefits of glideInWMS are preserved excepting prioritization/fairshare
- Would enable PanDA to explore Condor fairshare mechanisms progressively
- Recent proposal, not completely thought through by us
- A certificate is bound to a user temporarily. Ideally you would like all pilots to accept all users' jobs

Conclusions

Conclusions: PanDA server

- Main differences between PanDA server and CMS analysis framework
  - Complexity of the systems and levels of queuing

	Architecture	Upside	Downside
PanDA	Simple central architecture	Global view and control	Potential single point of failure
CMS WMS	Distributed 2-level queuing	Higher scalability & reliability	No global view

  - Resource allocation
    - Dynamic brokerage in PanDA, more fixed in CMS WMSystem given distributed character
- PanDA server is modular
  - Classes for externals (e.g. DM) can be implemented easily
- Coupling between WM and DM is not extremely tight
  - CMS I/O data handling could be handled

Conclusions: Pilots

- PanDA Pilot:
  - Rather ATLAS specific, but modularization is being planned
  - Proof-of-concept for CMS would be possible today
- AutoPyFactory:
  - Modular architecture where almost everything is pluggable
- GlideInWMS:
  - Attractive service where we can find common ground
  - Various scenarios for plugging GlideInWMS to the VO job management systems
    - Scenario A is a pragmatic approach to start working
    - However we need to think more about the different possibilities

Future work

- **Next step in the Feasibility Study:**
  - User workflows, error troubleshooting and monitoring
- **If we convince ourselves about the feasibility:**
  - Define a detailed proposal for a proof-of-concept
  - We would like to try a “Hello World” as a further check

Acknowledgements

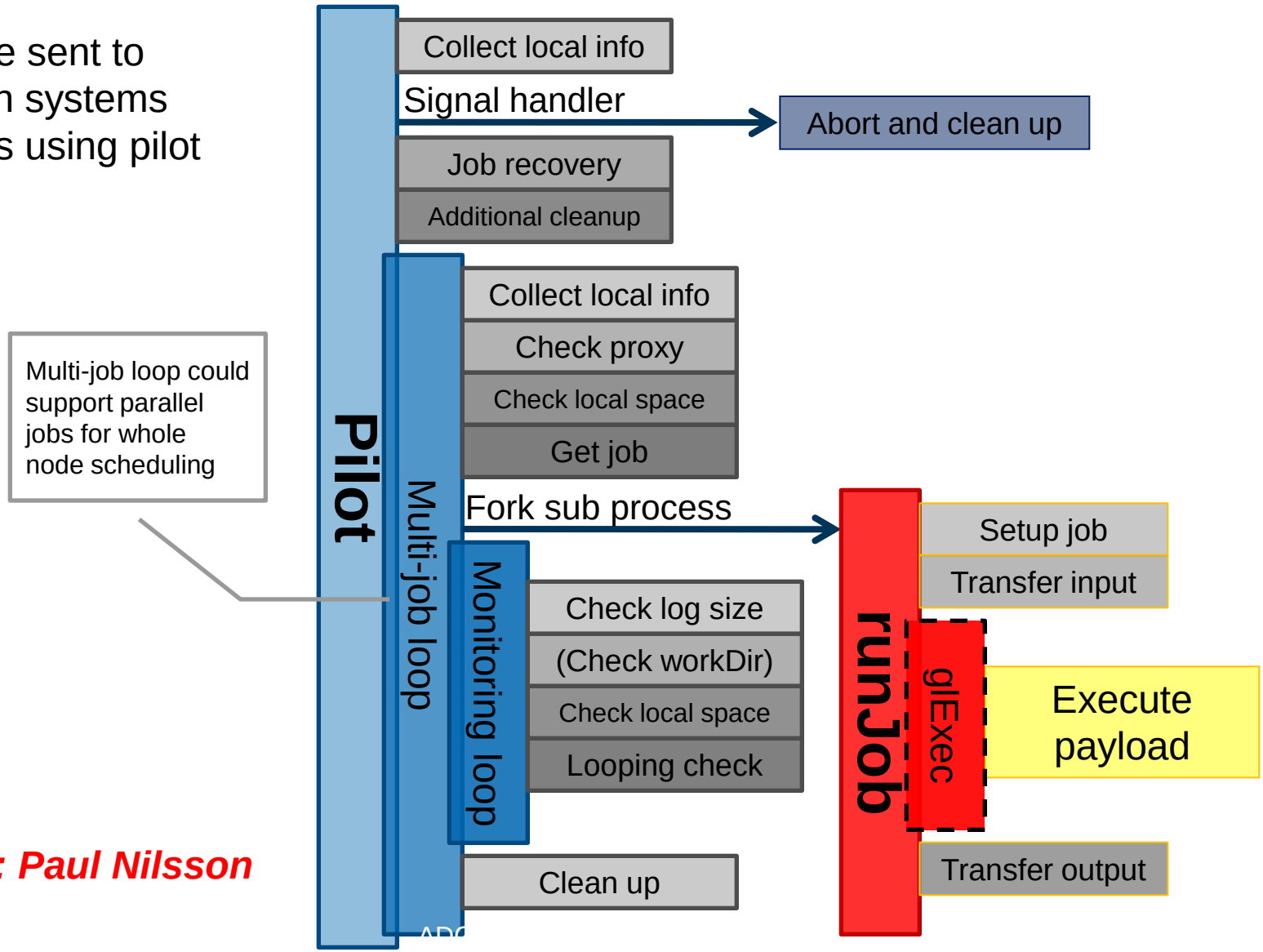
We are very thankful to the experts for their time and ideas

- Jose Caballero
- Simone Campana
- Burt Holzman
- John Hover
- Steve Foulkes
- Claudio Grandi
- Tadashi Maeno
- Paul Nilsson
- Maxim Potekhin
- Igor Sfiligoi
- Eric Vaandering
- Rodney Walker
- Torre Wenaus

Backup

Panda Pilot Overview

Pilots are sent to the batch systems and WNs using pilot factories



Credits: Paul Nilsson