Exposing Federated Datastores to Users

Brian Bockelman
XRootD Workshop @ UCSD
January 2015

Preface

- The original title was “Historical Perspective & Future Direction of CRAB3”
  - CRAB3 is the new version of computing tool for CMS users to run analysis tasks.

- However, that’s the *solution* for running analysis. Let’s focus on the *question:*

  - *How do CMS users access data?*

The Basics of CMS Data Access

- For about the 8 years, CMS has had these core concepts:
  - Logical File Name (LFN): The name of a known file within the experiment. Looks like `/store/data/foo`.
  - Physical File Name (PFN): A URL where a given LFN can be accessed. Looks like root://xrd.example.com/store/data/foo.
  - Trivial File Catalog (TFC): An ordered list of of regular expressions that, given a protocol and LFN, will produce a PFN.
  - Each storage site is operated independently with its own namespace. In the job's runtime environment, the TFC is found in a fixed location. The user specifies a LFN, the CMSSW application translates to a PFN, and the resulting URL is fed to ROOT.

Data Access in the Stone Age

- In the Stone Age (pre-2009), how did CMS users access a single event?
  - Before anything else, use DBS to figure out which file held the event of interest.
  - **Submit a job**: (The Right Way ©). Use the CMS Remote Analysis Builder (CRAB).
    - Write a CRAB task description to analyze a file.
    - Write a CMSSW config file to filter out the interesting event and save it to disk.
    - Submit to glite. Wait. Wait. Resubmit. Wait. Wait. Hope the output is there.
  - **OR build your own**: PhEDEx provided enough information to figure out a SRM URL for the file. Write a tool to query PhEDEx, then run “srmcp” by hand.
    - Wait. Get error. Fix. Wait. Failed transfer. Email experts. Hope someone sends you file out of pity.

Data Access At A Site

- The runtime configuration information for a job is found in the directory **/cvmfs/cms.cern.ch/SITECONF/local**.
  - This is a “magic symlink” in CVMFS, controlled by the site.
  - Files in this directory specify the default protocol, TFC, available HTTP proxies, site name, etc.
- **Important**: the interface between central CMS and the site is the LFN; no lookup services or central databases are needed.
  - This was critical - sites could toy with data access services without involving central CMS. Without these experiments, we would still be in the stone age.

First At Bat - FileMover

- Needless to say, the Stone Age was not fun!
- In 2009, I started my first attempt at federating data stores, **FileMover**.
  - Basically, I implemented the "**Build Your Own**" approach from the previous slide in a webapp.
    - User logged into webapp, requesting a file or event.
    - Webapp located the file, executed *srmcp* (and maybe CMSSW) to download to a 1TB disk on webserver.
    - Once *srmcp* complete, user got a download link.
    - Some niceties: Provided a progress bar so you can see the system was working, had built-in rate limits (5 files per user).

Strikeout - FileMover

- FileMover was lightly used - never truly popular. Why?
    - **Slow**: The central web server had to download the whole file; you downloaded the file to your laptop, then had to scp it to your development environment.
        - Somewhat better if you used the single-event mode.
    - **Errors exposed**: Limited error recovery; hard to translate between SRM errors and something for human consumption; hard to provide UI for files on tape. “Is it broken or is it stuck?”
    - **Different environment**: Users had to switch between their remote login/terminal and the laptop/web.
    - **Tiny scale**: Allowed developers to debug single events or single files. Too painful to do even toy analyses.
- We did some R&D for FileMover V2: using a C API for SRM and GridFTP, provide a FUSE filesystem that allowed random-ish reads. Chunks of files were downloaded into a cache on demand. Plan was to export the filesystem via web server and Xrootd.
    - Technically difficult. May be more plausible today - GFAL2 is much better than what was available at the time.
    - Didn’t fix the scale or “errors exposed” issue above. Much better speed and accessible via ROOT.

In the meantime: Watershed Year - 2009

- When experimenting with FileMover V2 in early 2009, I got acquainted with the Xrootd server.
- In parallel, the Nebraska Tier-2 started experimenting with HDFS (Hadoop 0.18) in October 2008.
    - In February 2009, Hadoop-0.19.1 was released, with the last feature we needed to try the filesystem at scale (user permissions).
    - By March 2009, we started converting the Tier-2 site storage to HDFS. Within two months, three other Tier-2's started converting.
- Random read performance for HDFS 0.19 was somewhat poor - I started work on decreasing random reads in CMS around February 2009. This led to the work in my next talk.
- Due to lack of internal security, HDFS could not be exported outside our cluster. However, HDFS gave us a thread-safe C API for our storage system.
    - In June 2009, started HDFS integration with the Xrootd server: this allowed on-site users to access HDFS from outside the cluster.
    - In October 2009, integrated support for the TFC.

What is needed?

- What's needed in a federated data store to be successful?
    - It must actually be a federated data store: Uniform namespace, Independent storage, transparently accessible.
    - Client must be robust against failures. Unless necessary, failures at a single site shouldn't .
    - The experience must be similar to local data access.
        - Data should be available quickly - the first byte should arrive in seconds.
        - The performance hit should be minimal.
        - The federation must be supported by all user applications.
    - All interesting data should be accessible. That means we want most sites participate. The system should be low-maintenance, not crash the site, and be popular with local physicists.

A Federation is Born!

- At the end of 2009 / early 2010, we setup a redirector at Nebraska and got Caltech and UCSD to join the federation.
    - It took about a year to get the remaining US sites to participate.
        - Things worked well at the scale; US sites have fairly even performance and we are a tight-knit community when problems did occur.
- In 2011, the AAA collaboration (Nebraska, UCSD, Wisconsin) received funding to grow the federation to LHC scale. Improve operations, manage capacity, improve monitoring, effective integration with CMSSW, integrate with workflow management.

User Interaction

- What can a user do with this federation?
    - Open any file at the participating sites knowing only the LFN.
        - Using CMSSW or bare ROOT.
        - Or just download it with *xrdcp*.
- Compared to FileMover:
    - **Not slow**: As we only included files on disk, response time was acceptable.
    - **Same environment**: All the tools users know and love.
    - **Much larger scale**: The system scaled with the amount of hardware deployed.
    - **Acceptable error handling**: The Xrootd client had robust error handling, although we wanted it to be more aggressive.
- At first, this existed in a separate world from any official tools.

CRAB Primer

- A few core concepts:
  - Given a user sandbox, an input dataset, and a few splitting parameters, CRAB creates a corresponding bag of jobs.
  - CRAB allows users to perform job management tasks (kill / resubmit / status).
  - And is responsible for returning output files to user (optionally registering the output dataset).

CRAB2 Support, V1

- CRAB2 is a client-side tool: all state and code is distributed as a client.
    - The central components are simply a HTCondor submit host.
- Thus, our first attempt at support was a CRAB2 patch. Users could “flip a switch” and send jobs anywhere in the US as long the file was at one of three US Tier-2s.
    - Patch never made it into the official release. Had issues like the fact that participating sites were hardcoded.

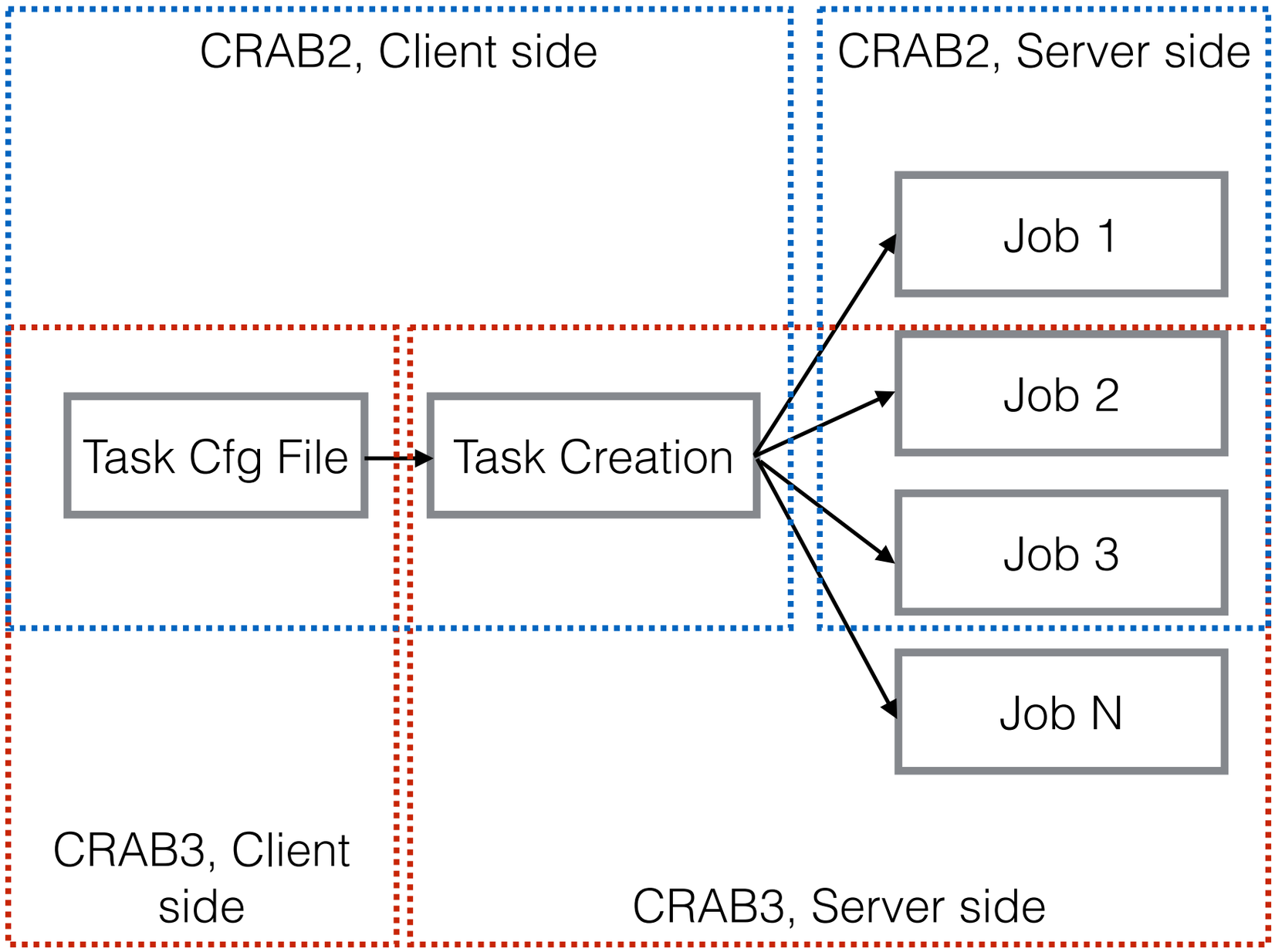
CRAB2 Support, V2 - GlideinWMS Overflows

- glideinWMS is the software we use for resource provisioning.
    - Think of it as HaaS - HTCondor as a Service. Given a HTCondor pool with idle jobs, it is tasked to create worker nodes through that pool. Start VMs, send pilots, etc.
- For idle jobs in queue for more than 24 hours, we would “fake” pilot jobs: if the idle job could run at Nebraska, we may send a pilot to Purdue claiming to be running at Nebraska.
- Compared to the previous client configuration knob:
    - We could centrally limit the number of pilots running overflows.
    - Users had no say in this: could not opt-out or opt-in to overflows.

CRAB3

- CRAB2 has a few warts:
    - No automation of job retries. As CRAB2 targeted multiple job backends (gLite, HTCondor, PBS, etc), we had the least common denominator of capabilities.
    - User output files are transferred from worker node to user's remote SE: not an optimal data path.
        - "Heavyweight" client contained all the logic: impossible to push out changes or fixes without a client upgrade.
- CRAB3 set out to fix these:
    - Client/server architecture: CRAB3 client uploads a task description centrally. Central CRAB3 components handle the job splitting and other job prep, then submit to a backend.
    - User files are stageout out to runtime site's local SE. A new component, Asynchronous Stage Out (ASO), moved files from execution site to final destination.
- CRAB2 tracked individual **jobs in a job backend**; CRAB3 tracks **tasks in a task backend**.

CRAB3 vs CRAB2 - Job Creation



CRAB3 Backends

- Through its life, CRAB3 has had four backends:
  - **WMAgent**: Same processing layer as CMS Production team.
  - **PanDA**: Pilot-based workflow management system; best known as ATLAS's
  - **HTCondor/DAGMan**: Well-known workflow management system from Wisconsin. Can be integrated with glideinWMS. Currently used in production.
  - **BOINC**: Developed by CERN IT's volunteer computing effort.
- Due to limited effort, CMS has only been able to keep a single backend in production at a time. As the product matures, we may be able to revive support for others.

CRAB3 and Federations

- CRAB3 has an “ignore locality” switch built-in. If enabled, the jobs will only obey the user-specified white/blacklist — not the data locality!
  - This comes with a downside: access via data federation, almost by definition, will have a performance and reliability hit. *Our job is to shrink that hit!*

- CRAB3 also allows users to specify an arbitrary list of files as input - not necessarily a CMS dataset. Can refer to the federation directly.

CRAB3 - Future

- Currently, “ignore_locality” is an on/off switch. I want to make this on/maybe/off.
    - Although it hasn’t been a problem yet, some day a user will abuse this power. “Emergency lanes on the highway don’t work if everyone tries to drive in them.”
    - We are working to migrate the overflow setup into the glideinWMS instance for CRAB3. The user could specify:
        - ON: Ignore data locality for this job.
        - MAYBE: Ignore data locality if this job has been in queue for more than 24 hours.
        - OFF: Never ignore data locality.
        - I think it would be useful for both the analysis and production side of the house!
- In addition, we could control the total number of “ignore locality” jobs centrally to prevent abuse.

Lessons Learned

- For an experiment just starting up, user data access plays second-fiddle to concerns like simulation or data processing.
    - Perhaps rightly so - however, this is where the “unwashed masses” will actually interact with computing.
    - Investments here pay off! Otherwise, impediments to access means professors have to “waste” a grad student on computing.
- Through foresight or luck, CMS computing was flexible enough so good ideas could come from outside the central organization and be successful.
    - Enough components were under site control that new ideas could start at a single site.
    - Eventually, the data federation could be incorporated back into the computing model - as evidenced by CRAB3.

Federation of the Future

- As HTTP integration with the Xrootd software matures, CMS will probably start investigating opening up the federation to other protocols.
    - Deep integration with CMSSW is likely not going to happen until Run3 activity due to the need for an intelligent HTTP client. While we have fully control over the CMSSW IO layer (see my next talk), we never change the software post-release & must support old versions.
- One thing we miss - but was present in FileMover - is an event delivery service.
    - I'd like to see a few xrootd servers deployed around the world which can deliver individual events to clients.

Federation of the Future - Xrootd + HTCondor

- I see a need for tighter integration between the WMS and the data federation.
    - HTCondor is the common layer for all offline processing tasks (Tier-0 included!). Effectively, it is *the* resource information service for CMS.
        - I believe we should be advertising info about each Xrootd server in the CMS HTCondor collectors.
- To work outside the CMS context, we need to seriously work on the security model - users must never see grid certificates!
    - ALICE has had a token-based solution for this for approximately a decade. One path forward may be to leverage the HTCondor security infrastructure to create access tokens for Xrootd.

Some final thoughts

- Many things had to align to make AAA work:
    - Computing model already amenable to data federations.
    - Physics software / data model had to change.
    - Well-run, collaborating sites.
        - Effort provided by NSF.
        - Plus other factors - and some amount of luck!
- It also didn't happen overnight - direct efforts started 5 years ago.
    - This is a timescale consistent with other large changes to computing models.
      New ideas for LHC Run 3 should have been started last year!
- Repeating the accomplishment would be take less effort for another HEP collaboration, although the same fundamentals need to be in place.
- In the end, I think we achieved our underlying goal: improving data access for HEP.

Questions?