StashCache: Data Services for the OSG

Brian Bockelman,
On Behalf of the StashCache Team

Background

- The two large LHC VOs, ATLAS and CMS, own storage at many OSG sites and use them as **storage elements**, or remotely accessible file systems.
    - These SEs behave like - and are operated like - POSIX filesystems.
        - For each POSIX command (cp, ls, mv, rm), there is an equivalent command for the SE. For the SRM protocol, for example, srmcp, srmls, srmmv, srmrm.
- The SE abstraction is **very low level**!
    - Managing data is analogous to having a login to 50 clusters.
    - Or copying files manually between your work desktop, laptop, phone, and home desktop.

Background

- How is data handled in the SE paradigm?
    - *Access*: Each SE has its own twist on data access. Either hardcode access rules locally (**yuck**!) or come up with a standard site discovery mechanism (**far less successful** than hardcoding!).
    - *Movement*: A service is given a set of files from endpoint A to endpoint B. The files are usable once files are at endpoint B.
    - *Catalogue*: Some central service tracks the location of each file.
        - Catalogs must be kept in sync for this to work!
    - *Data management*: Rules engine verifies that all files are in the “correct” location according to some set of rules. If not, make new copies with the movement service.
- Data lost? Site initiates a recovery procedure. In CMS, the site admin opens a ticket.
    - It is assumed this is an *exceptional event* which does not happen frequently.
    - If a file is not in the correct location, it can be considered an error.

Motivation

Opportunistic Computing is like giving away empty airline seats; the plane was going to fly regardless.

Opportunistic Storage is like giving away real estate.

(paraphrased from Mike Norman)

Motivation

- Using the SE paradigm has been a ***colossal failure*** for opportunistic VOs.
    - Systems for CMS and ATLAS are robust and efficient, but proven impossible for others. Cost of management is too high and opportunistic VOs are unable to command site admin time.
- Key to this failure is the underlying assumption in the SE paradigm that file loss is an exceptional event.
    - Again, “Storage is like real estate.”
    - To be successful, opportunistic storage must treat file loss as a *everyday, expected occurrence.*
- The lack of high-speed local storage **significantly decreases the range of workflows opportunistic** VOs can run on the OSG.

A Different Paradigm: Caching

- A file is downloaded locally to the cache from an **origin server** on first access.
    - On future accesses, the **local copy** is used.
    - When more room needs to be made for access, **“old” files are removed** (by some algorithm which decides the definition of “old”).
- Downsides:
    - Caching is only useful is the **working set size** is less than the cache size.
        - Otherwise, the system performance is limited to the bandwidth of the system feeding the cache.
        - Working set size is difficult to estimate for multi-VO.
    - **Not all workflows are supported**. This does not work well if files need to be modified.

Hypothesis: A significant number of opportunistic workflows have cache-friendly access patterns

Why Caching?

- Compare to caching:
  - *Access*: All endpoints in infrastructure have same data access method.
  - *Movement*: If files are not local, they are moved in on-demand.
  - *Catalogue*: All files are assumed to be at the “origin server”. We do not need to track any other location information.
  - *Data management*: Custodial copy of all files are at the origin; no other explicit work is needed by VO.
- More resilient against failures, less work to do. Sites can reclaim storage at any time (or other users can take it!)

Where Do We Use Caching Today?

- Most sites have a local HTTP cache for for the Frontier application and/or CVMFS.
  Why not use that?

  - HTTP cache deploys have been sized to match the ~1GB working set size of these use cases.

    - Caches are typically sized/restricted to serve the local site.

- Our target is 5GB-5TB working set sizes for tasks.

- We'd also like to have ***zero*** service requirements for the local sites.

  - CVMFS can have a light footprint, but it's not zero.

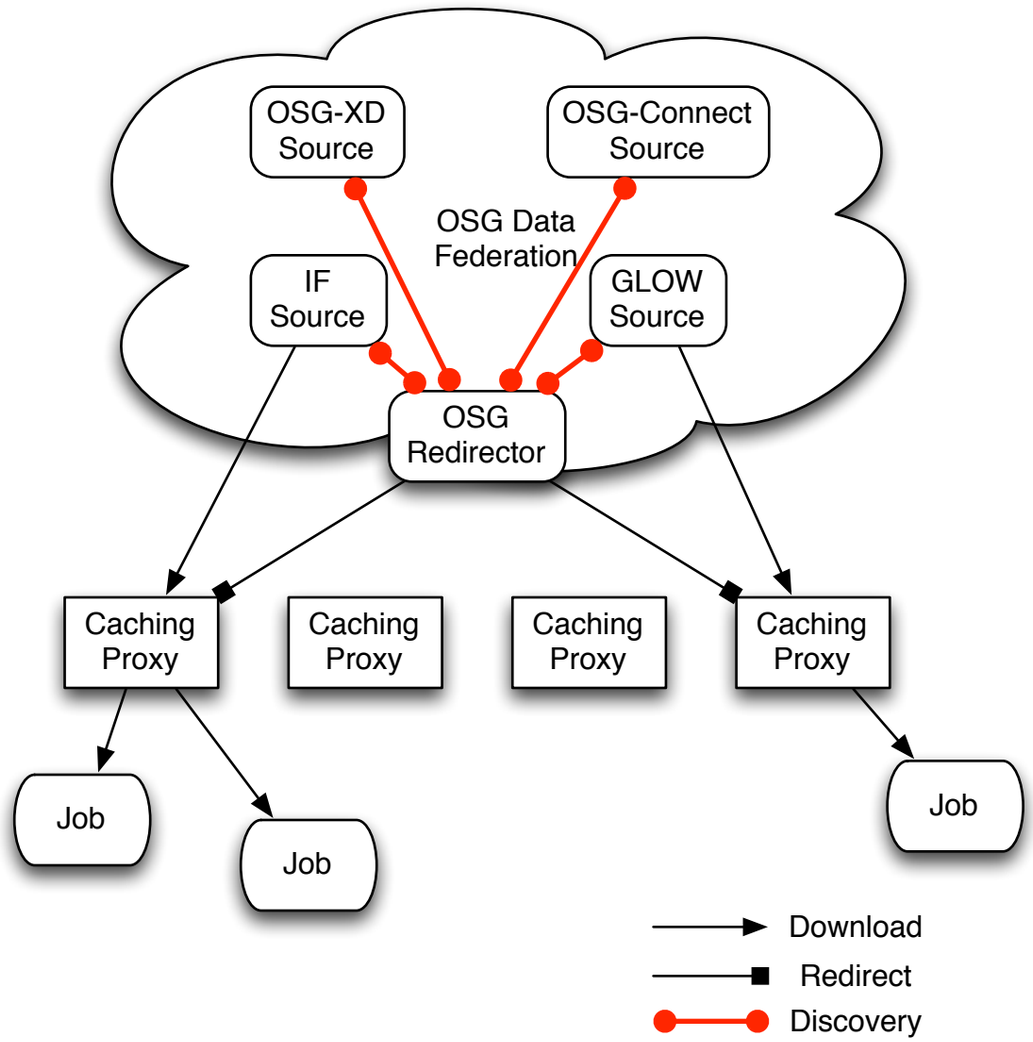
  - Further, we need to redesign caches for much larger working set size.

Big Idea:

Can OSG provide a caching service for opportunistic VOs?

Introducing StashCache

- Caching infrastructure based on SLAC Xrootd server & xrootd protocol.
- Each VO has a origin server.
- Cache servers are placed at several strategic cache locations across the OSG.
- Jobs utilize “nearby” cache, for some definition of nearby.



Original Architecture

StashCache locations & compute sites

Stash origin: ★

OSG

Caches: ●

Slide by Anna Olson (https://indico.cern.ch/event/330212/session/6/contribution/31/material/slides/0.pdf)

Scale and Scope

- The possible origin servers are limited to OSG VOs.
    - During the pilot phase, we have a single origin server (OSG-Connect).
- **No service requirement for each site.**
- Each cache has minimum size (>10TB) and performance (10Gbps to WAN).
    - This allows us to provide reasonable lower bounds on acceptable working set size.
- Scale system so it can support ~10k running jobs.
- Scope of the system is limited to data stage-in, not stage-out.

Data Access Methods

- Xrootd is not a familiar protocol for users. Goal is to provide reasonable UIs to VOs; users don't care about protocols, they care about interfaces.
    - This means application protocol is implementation-defined; if protocol B is more relevant in 3 years, we can use that.
    - C.f., accessing "google.com" from Chrome does not use HTTP. Few users seem to care as long as the browser (the interface) works.
- To upload files, VOs can provide users with a writeable shared filesystem exported by the origin server.
    - Users first must "cp" their data to this mount point, then can access the files from their jobs.
    - Top-level directory name is assigned to VO by OSG; VO manages the namespace within their directory.
- User interfaces:
    - "cp"-like
    - HTCondor file transfer
    - POSIX

“cp”-like

- All glideins are instrumented with to have “stashcp” in the $PATH.
- stashcp emulates the CLI of venerable POSIX “cp”.
- Users simply say:
  - stashcp stash:/user/bbockelm/foo $PWD
- Note no implementation details exposed!
- Summary of use statistics, performance, and errors encountered are injected back to HTCondor ClassAd.
  - All Stash usage becomes query-able with condor_history.

HTCondor File Transfer

- The workflow system (here, HTCondor) can manage file transfers directly.
    - The pilot configuration provides a callout script for handling a given URL type. Underneath, this is implemented using stashcp.
        - Using HTCondor file transfer plugins provides a mechanism for concurrency management, policy-based retries, and removes need for error handling in user jobs.
            - HTCondor understands “file transfer failed” semantics directly.
- Users add the following line to their JDL:
    - transfer_input_files = stash://user/bbockelm/foo

POSIX

- ‘stashcp’ and HTCondor file transfer plugins require the entire file to be downloaded locally.
    - Not all worker nodes have large enough scratch disk.
- These ‘cp’ like interfaces can be difficult for applications which do not know what files will be read - or require complex directory structures.
- Using a LD_PRELOAD library from the Xrootd team, we can make StashCache appear to be a POSIX filesystem to the application.
    - As many applications perform small reads, it uses the local filesystem as a cache for accessed portions of the file.
    - All “normal” POSIX utilities and APIs will work (think “ls”, “cat”, “tail”, etc).
- Simply set “+UseStashCachePosix=true” in the HTCondor submit file.
    - LD_PRELOAD can have some overhead and may not work in all cases; hence, users must explicitly ask for it.

Operations

- The StashCache service has a few basic components:
  - **Origin servers**: one per VO. Run by the VO.
  - **Redirector**: one for the entire system. Run by OSG Operations.
  - **Cache server**: 5-10 for all of OSG. Run by ???
    - Looks like a site service (so, run by site admins) but behaves like a central service (shared amongst several sites). Characteristics of a both a Stratum-1 and site squid in the CVMFS ecosystem.
    - We're aggressively looking at adding more remote debugging and restart capabilities than a typical OSG service.
      Ultimately, host site is always responsible for hardware and OS basics.
- In addition, there's various pilot-side software. Distributed via CVMFS.

What's Real?

- The StashCache system has been tested by the OSG-Connect team for several months.
  - Limitation is the cache servers connect to the origin directly.
- Currently adding in the redirector so we have the *capability* to have additional origin servers - even if we keep the existing OSG-Connect server.
- Current timeline is to open to external OSG VOs around May.
  - Will be looking for bleeding edge users. Expect a long testing period before we declare production.

Future / Deferred Work

- Plenty of work in the short term:
    - Improve remote debugging / management of cache servers.
    - Add monitoring of cache health and performance.
    - Provide non-CVMFS distribution of software in pilots.
        - Operate, package, debug, understand.
- We've left out a key piece: cache management.
    - Currently, plan on working closely with users to make sure they understand the working set size limitations.
        - Minimize problem by having cache sizes in 10s of TB.
        - Monitor for new problematic workflows.
    - Long-term, want to invest in technologies that avoid cache thrashing through pinning of datasets. C.f. Derek Weitzel's dissertation work with condor_cached.

Parting Shots

- The SE paradigm provides a low-level interface to storage, allowing VOs to customize every detail of their data management.
    - This works out poorly for opportunistic sites.
- StashCache implements a cache-based data management paradigm; applicable to many workflows for opportunistic VOs.
    - StashCache targets datasets in range 5GB-5TB.
    - Service is run by VOs, Ops, and 5-10 host sites. No new service at the average OSG site. No new software to install.
- In internal integration & testing now. Will be made available to additional VOs throughout the year if all goes well.

Questions? Thoughts? Opinions?

- For more detailed info, see Anna Olson's presentation at the UCSD XRootD workshop:
  - https://indico.cern.ch/event/330212/session/6/contribution/31/material/slides/0.pdf
- Credit where credit's due:
  - StashCache is a (very) modest extension of ideas and implementation originally done by the OSG Connect team.

Backup Slides

Slide Courtesy Anna Olson

Testing & measurement methods

- Send jobs out to OSG
- Each job pulls a number of files using either stashcp (for Stash Cache) or wget from STASH
  - Locations: BU, UChicago, STASH, UCSD
  - Files downloaded to either job sandbox or /dev/null
  - Single or multiple jobs sent out at a time
- Source, destination, file size, and download time are recorded

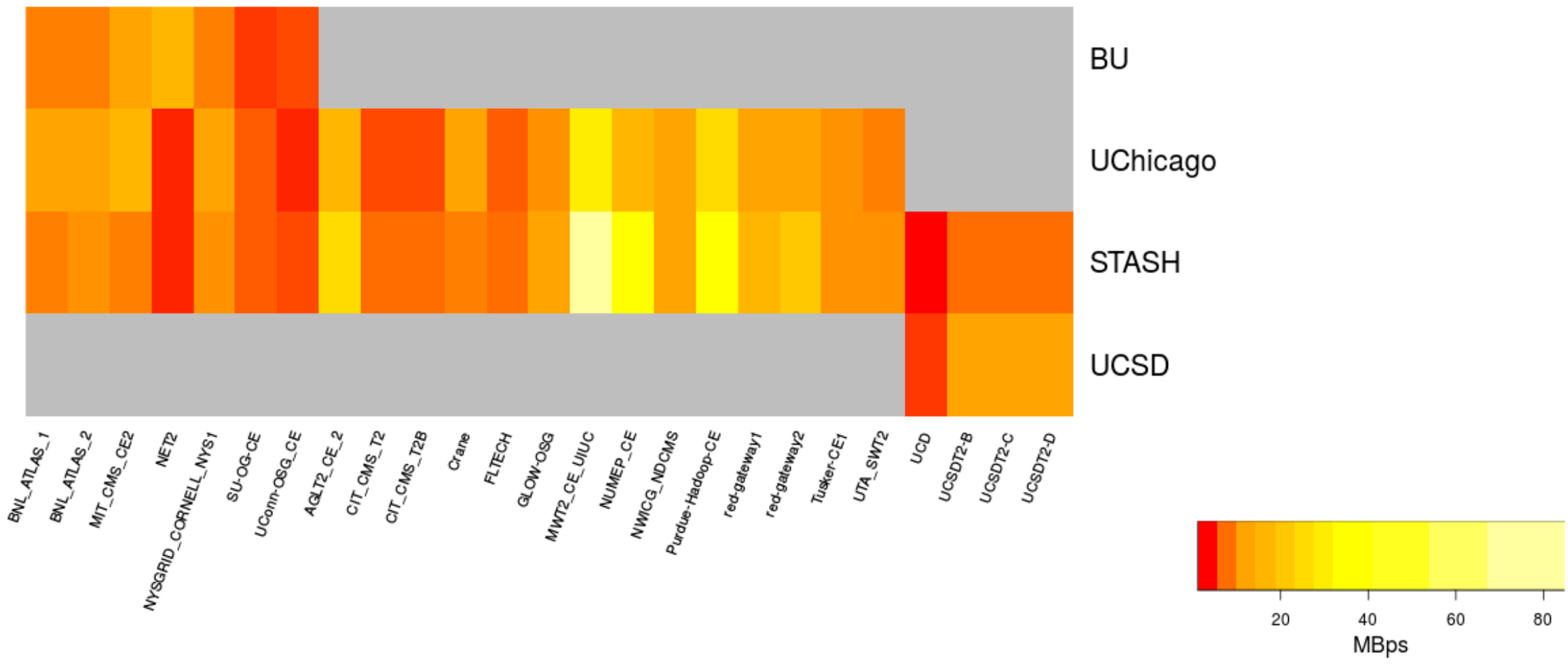
Slide Courtesy Anna Olson

Caching tests

- Pull same files multiple times
- 100 jobs sent out, each pulling 10 files in series
  - Multiple jobs could be pulling from the same source!
- Available sources: BU, UChicago, UCSD and STASH
- File size ranged from 750KB to 21GB

Slide Courtesy Anna Olson

Median download speed: all files



Slide Courtesy Anna Olson

Distribution of speeds: UCSD (as destination)