Data Management:

Prologue

Where is my data?
How do I access it?

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! No help from POSIX here! Either require specialized storage (limiting!) systems or live with the mess.
    - *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 **data loss 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, SDSC)

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** usefulness of the OSG for FIFE.

Motivation:
FIFE needs this

- Many FIFE workflows can exist blissfully with minimal data management:
    - “Never underestimate a condor_schedd with a 10Gbps interface and a nice RAID.”
        - Generally, the limit is when the average file transfer per job is >1GB.
- Some workflows (“flux files”-based MC generation) need multiple GB of input.
    - **If FIFE wants to use opportunistic OSG sites, they need a solution!**

Data Management in OSG: StashCache

Brian Bockelman

A Different Approach

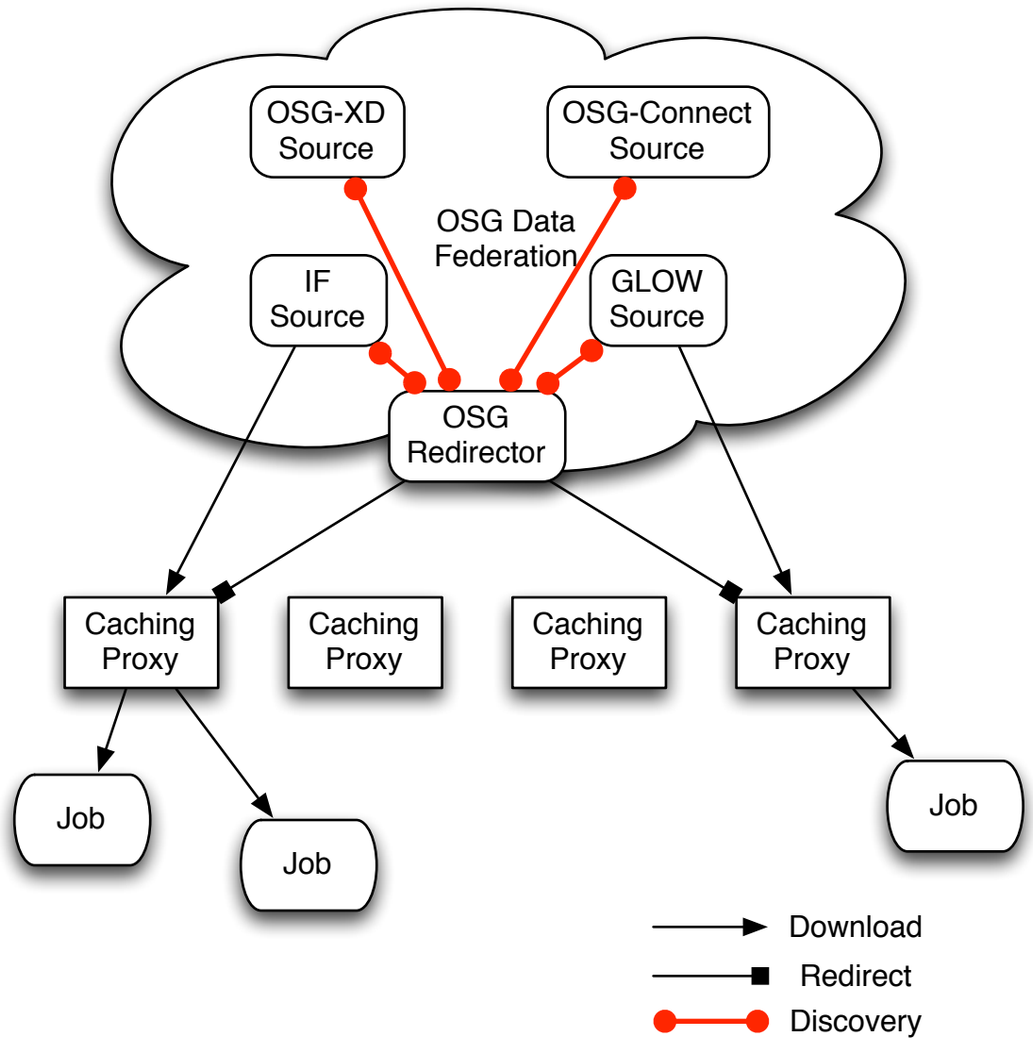
- *Caching*: A file is downloaded locally to the cache from an **origin server** on first access.
    - On future accesses, the **local copy** is used.
    - When more room needs to be made for access, **“old” files are removed** (by some algorithm which decides the definition of “old”).
- More resilient against failures, less work to do. Sites can reclaim storage at any time (or other users can take it!). **Data “loss” is normal** (loss == cache eviction).

Why Caching?

- Contrast with SEs:
  - *Access*: All endpoints in infrastructure have **same data access method**.
  - *Movement*: If files are not local, they are **moved 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.

Introducing StashCache

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



StashCache - Goals

- Provide effective, high-performance caching for working set sizes of 10GB-10TB.

- Require **no** special services or configuration for sites to participate.

- Provide high-quality access methods that abstract away underlying implementation.

StashCache - Data Access

- After user copies file to their VO's origin server (i.e., through mounted /pnfs for Fermilab VOs), Stash provides three data access methods:

  - **'cp'-like**: Can invoke `stashcp` from job wrapper to download files.

  - **HTCondor File Transfer**: HTCondor orchestrates transfer; list files needed in `condor_submit` file.

  - **POSIX-like**: Job wrapper accesses StashCache as if it were a mounted filesystem (limitations apply: uses LD_PRELOAD).

Use Case:
Flux file distribution for Nova.

- Test distributing “flux files”; assume they are O(1GB) and O(100) files.

- Each job reads 2 randomly-selected files from dataset.
  - Jobs last several hours.

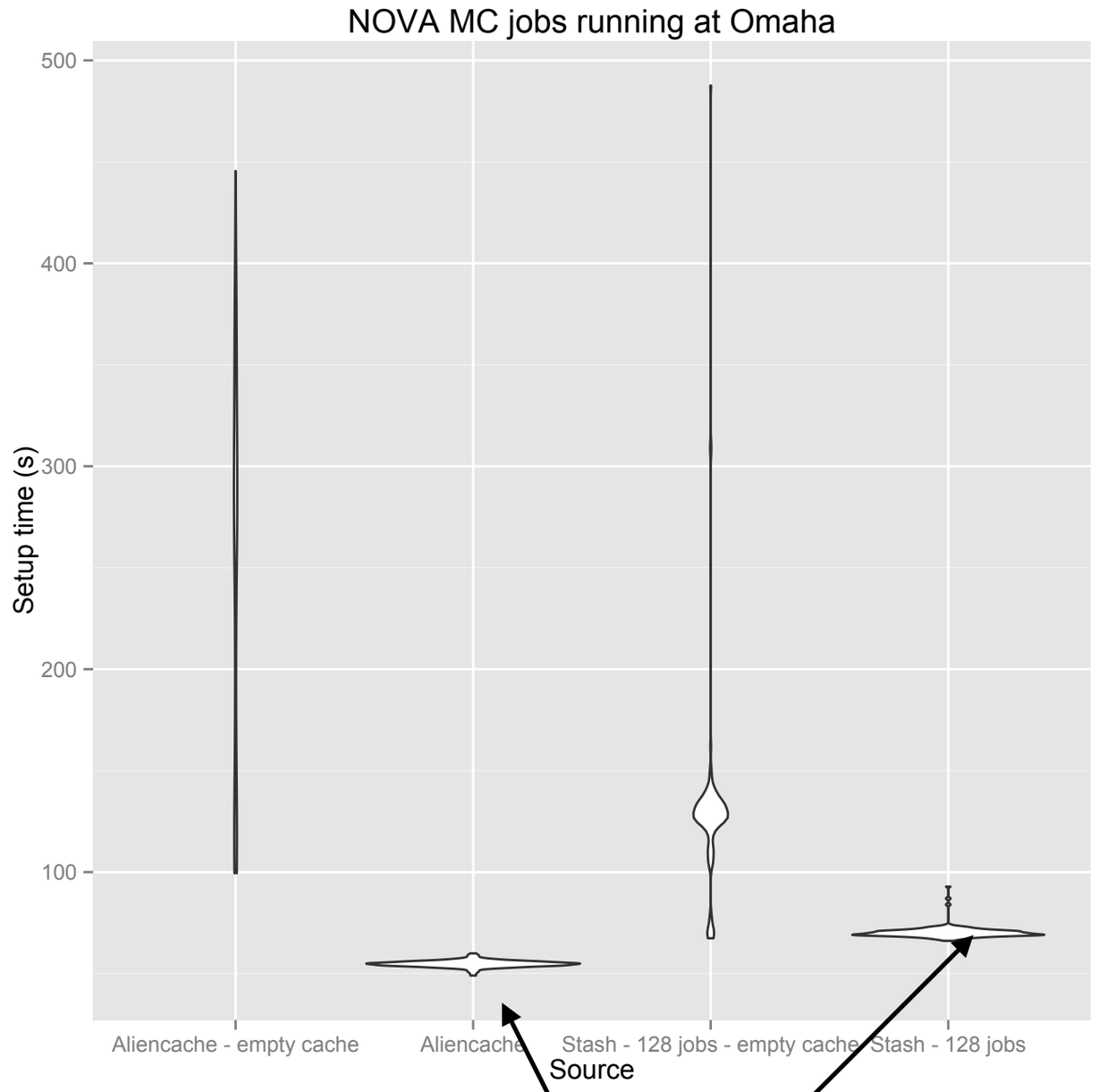
- Enough jobs are submitted so each file is read N times.

- Use the POSIX-like mode; minimum read size is 64MB to hide latency (file is buffered on worker node disk).

Evaluating StashCache for Nova.

- "Violin plot" to right shows distribution of startup times for 128 jobs using local SE (via aliencache) versus local cache.

- Tests done by Robert Illingworth using actual Nova framework



Cache hit - comparable performance

Nova use case - next steps

- Currently waiting on integrating FNAL dCache into stash as an origin server
    - (tests were performed by copying flux files to a different origin server).
- Would like to re-run tests tightly controlling the number of concurrently-running jobs.
- Explore larger scales: StashCache system aims to get up to 10,000 concurrent jobs.
    - Currently tests were on order of 128 jobs.

StashCache Futures

- We will be opening up StashCache to more users throughout the year.

- The real power in StashCache is the **distributed hardware**; as we go forward, will be experimenting with additional access modes. Current ideas:

  - Export **HTTP protocol** for more familiar client tools.

  - Integrate with the **condor_cached** (see Derek Weitzel's HTCondor Week 2015*) for space management.

  - **Use for distributing CVMFS repos**. Would allow aliencache-like performance without any site configuration or site services.

* http://research.cs.wisc.edu/htcondor/HTCondorWeek2015/presentations/WeitzelD_CacheDPres.pdf

Questions?