Controlled overflowing of data-intensive jobs from oversubscribed sites

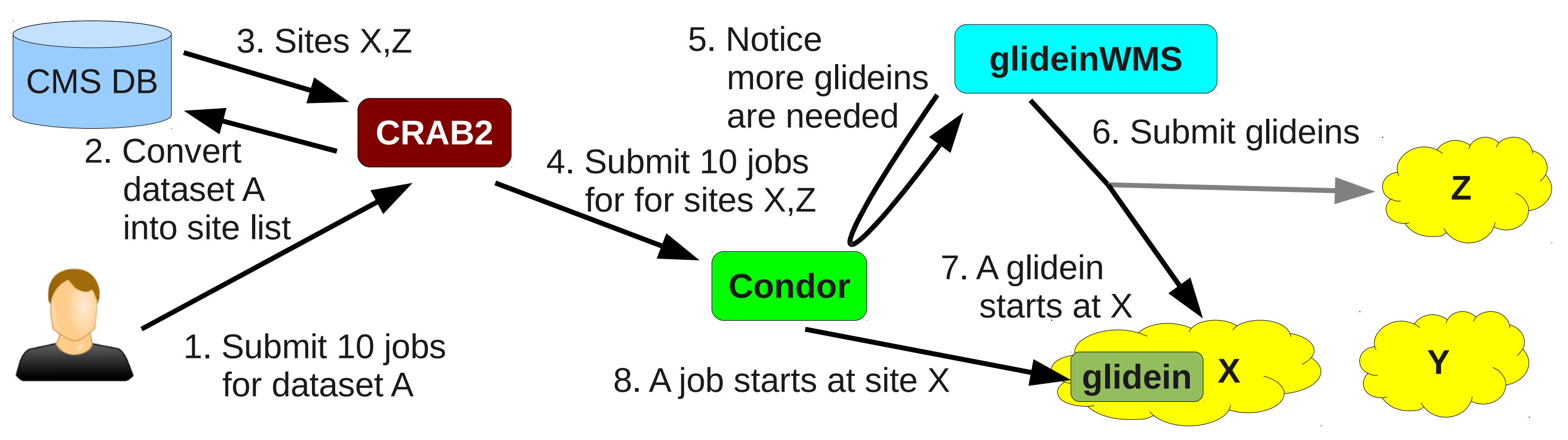
by
I Sfiligoi[1], F Wuerthwein[1], B Bockelman[2], D C Bradley[3], M Tadel[1], K Bloom[2], J Letts[1] and A Mrak Tadel[1]
[1]University of California San Diego, La Jolla, CA 92093, USA
[2]University of Nebraska – Lincoln, Lincoln, NE 68588, USA
[3]University of Wisconsin – Madison, Madison, WI 53706, USA

Open Science Grid

The CMS analysis computing model **was always relying on jobs running near the data**. This has the advantage of avoiding the use of Wide Area Networking (WAN), thus being limited by the throughput of the site-local storage subsystem only, and resulting in **high CPU utilization**.
However, it also restricts those same jobs to only CPUs close to the data. As a result, the proper placement of data at various sites is of paramount importance, and in CMS it was organized at management level, based on expected needs of the CMS community.

(See also talk #579 by A. Wakefield)

glideinWMS-based workflow shown


CMS DB
3. Sites X,Z
CRAB2
2. Convert dataset A into site list
1. Submit 10 jobs for dataset A
4. Submit 10 jobs for for sites X,Z
5. Notice more glideins are needed
glideinWMS
6. Submit glideins
Condor
7. A glidein starts at X
8. A job starts at site X
glidein
X
Y
Z

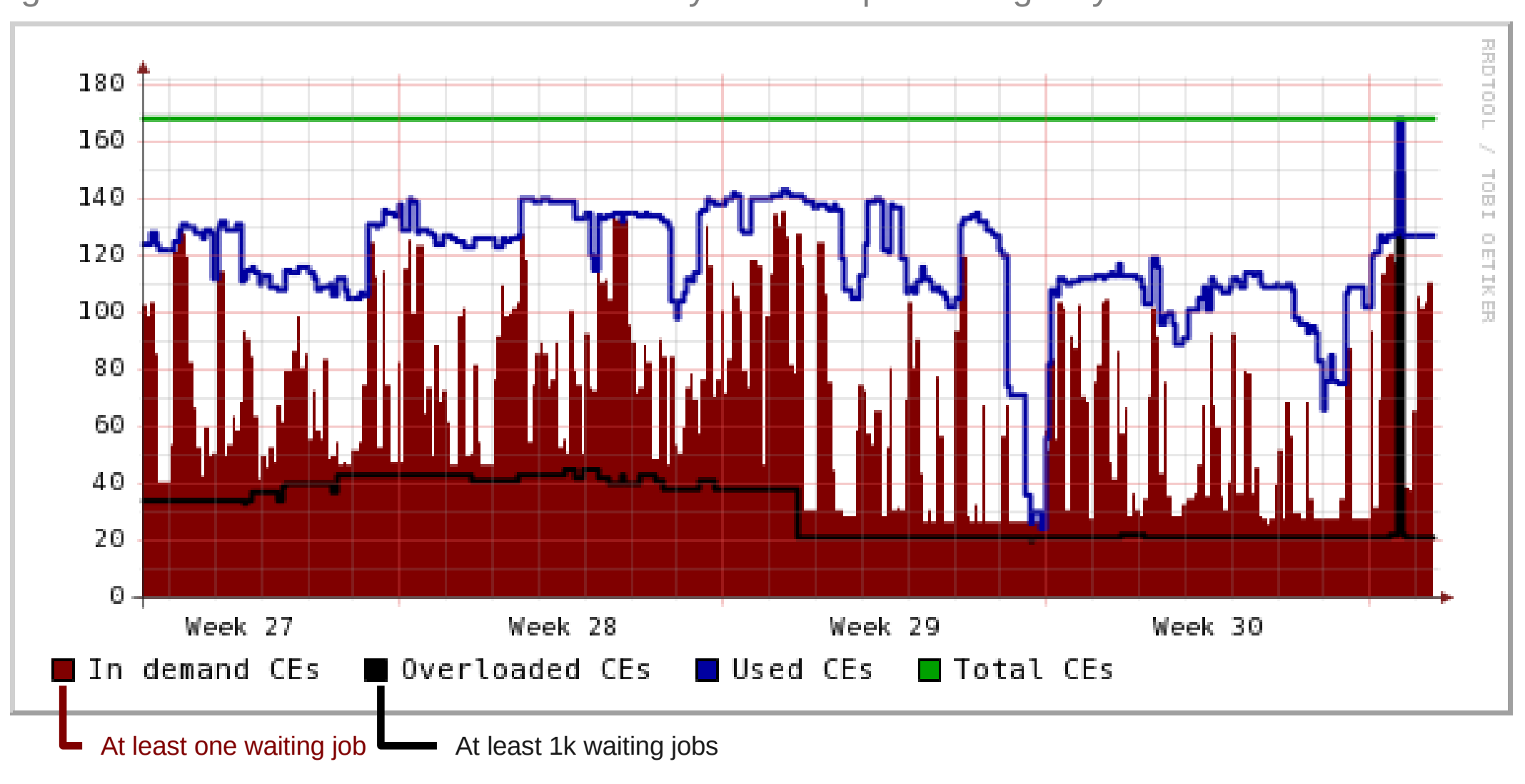
But there was still space for improvement.

CMS noticed periods of time when a large fraction of CPUs at certain sites were sitting idle due to lack of demand, while a large number of users' jobs were waiting for CPU at sites with popular physics streams. As a side effect, Terabytes of disk space hosting unpopular physics streams were never accessed in the same period of time.

Number of accesses	0	1	10	100	1k	10k	100k	1M
Number of datasets	272	161	511	629	175	114	34	2
Fraction of datasets	14%	8%	27%	33%	9.2%	6.0%	1.8%	0.11%

Above are the access statistics of the CMS datasets in use during summer 2011. As can be seen from the above table, 14% of all datasets were never used, while 2% of the datasets accounted for the vast majority of all activity.

glideinWMS-based instance at UCSD only - thus representing only a fraction of CMS use



Number of requested and actually used CMS CEs in the same period. As can be seen, only a subset is typically in demand or used. Moreover, only a few CEs are consistently in strong demand.

CMS thus decided to allow remote data access.

The chosen solution was to access the data through **real time Xrootd streaming** over WAN.
(See talk #381 by B. Bockelman)

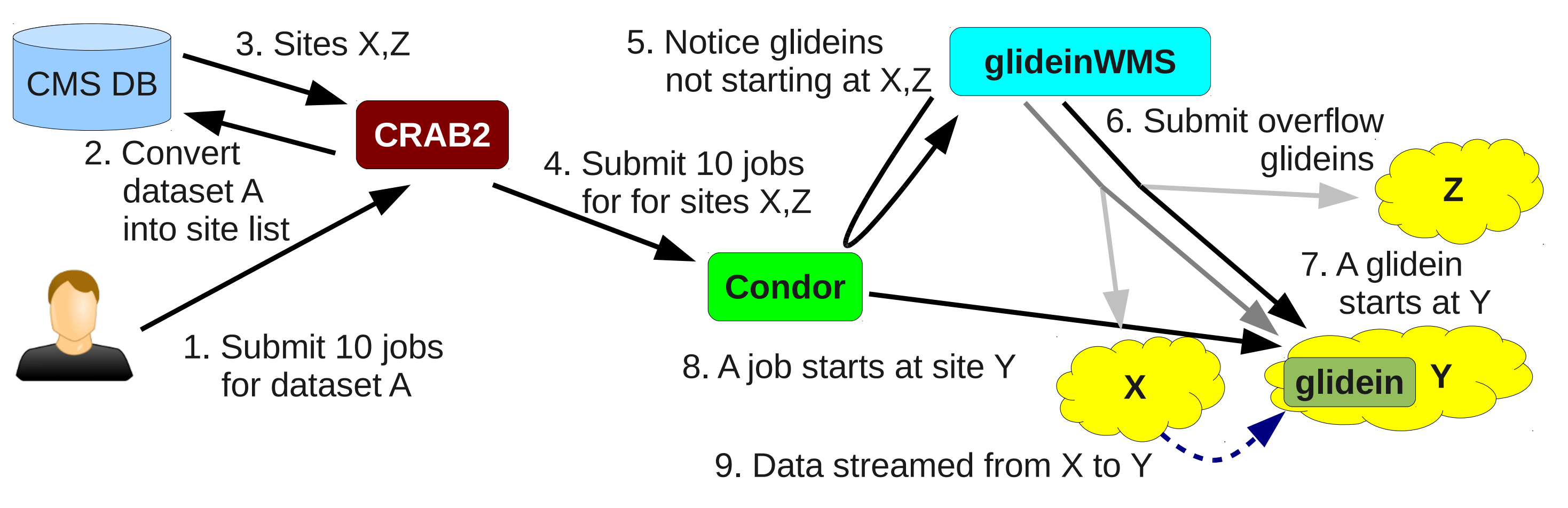
Reading data over WAN is obviously **less efficient than reading it from a site-local storage system**. However, **if there are no CMS jobs waiting for resources at one site, a slightly inefficient job is still orders of magnitude better than no job at all**. Said that, CMS has invested significant R&D effort into minimizing the inefficiency of WAN access, and recent CMS software can access remote data with **only about a 10% efficiency hit**.

both bytes/s and transactions/s

A potentially bigger problem is getting enough bandwidth out of the data hosting site. CMS sites size their storage systems based on the size of their compute cluster, which means that **the amount of bandwidth left for remote access is limited**. **If too many external processes were to start reading the hosted data**, it would likely drastically degrade the performance of the storage subsystem, reducing the efficiency of both the remote and the local processes. Nevertheless, most large CMS sites do have some spare bandwidth; it just must be carefully managed.

We need to control the remote access.

We changed the glideinWMS logic. The list of sites is used **just for regulating how many glideins** are being requested for each data source; jobs can now run anywhere. We have **limits on each source site**. We continue to also use the "regular logic" for most of the glideins, for efficiency reasons. The remote access should be used **only when the sites hosting the datasets are oversubscribed**, i.e. we want the **"overflow logic"**.


CMS DB
3. Sites X,Z
CRAB2
2. Convert dataset A into site list
1. Submit 10 jobs for dataset A
4. Submit 10 jobs for for sites X,Z
5. Notice glideins not starting at X,Z
glideinWMS
6. Submit overflow glideins
Condor
7. A glidein starts at Y
8. A job starts at site Y
9. Data streamed from X to Y
glidein
X
Y
Z

We estimate if a site is oversubscribed **by looking at the age of user jobs** in the Condor queues, where the existence of old jobs indicates that the site is not providing enough CPU resources. The current threshold has been set to 6 hours. Furthermore, when the overflow glideins are requested at other sites, overflow glideins should not "steal" CPU resources from regular glideins. We thus use a **lower priority credential** for overflow provisioning.

This required a change in matchmaking logic.

Originally we used the standard Condor practice of **expressing job requirements** as a boolean expression that was passed to Condor **during job submission**, with the glideins providing the needed attributes and essentially accepting any job.

Was impossible to mix the regular and overflow logics in the same system!

Job

Requirements=stringListMember(GLIDEIN_Site,
"UCSD,Nebraska")

Glidein@UCSD

GLIDEIN_Site="UCSD"
Requirements=True

Only change needed to the CMS software

Job

DESIRED_Sites="UCSD,Nebraska"
Requirements=True

Regular Glidein@UCSD

Requirements=stringListMember("UCSD",DESIRED_Sites)

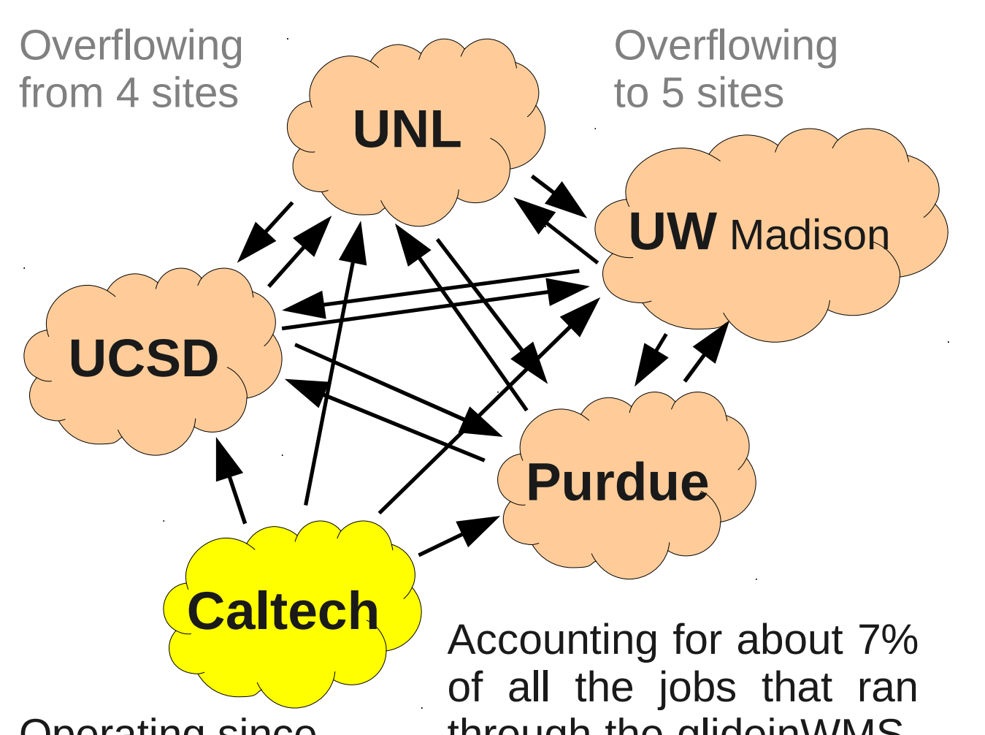
Overflow Glidein@UCSD

Requirements=stringlistmember("Nebraska",DESIRED_Sites)

We thus **moved the requirements entirely into the glideins**, with the **jobs just publishing appropriate attributes** and have no restrictions on where they can run.

Operational experience.

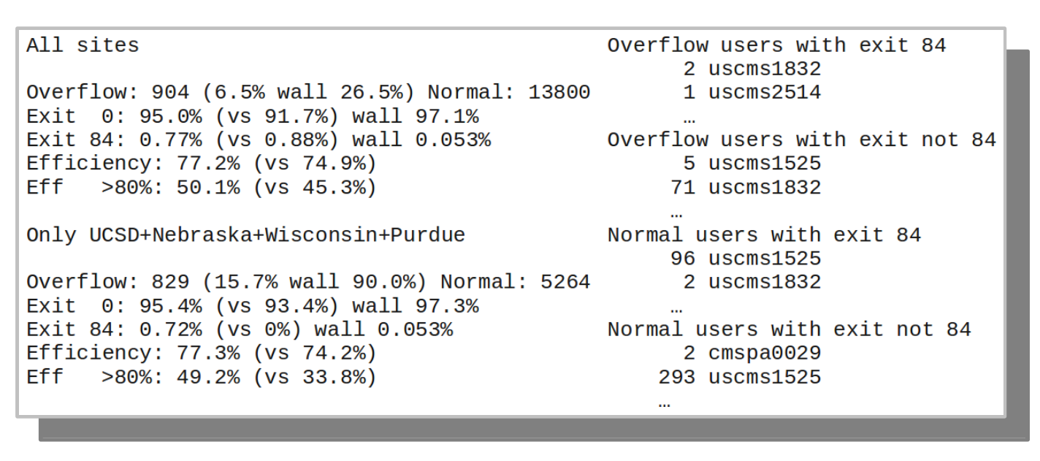
Still validating the concept, but **experience very positive** so far.



Operating since Fall 2011.

Accounting for about 7% of all the jobs that ran through the glideinWMS-based CRAB2 instance.

Monitoring shown to be essential to keep error rates low.



```
All sites                                        Overflow users with exit 84
                                                       2 uscms1832
Overflow: 904 (6.5% wall 26.5%) Normal: 13800          1 uscms2514
Exit  0: 95.0% (vs 91.7%) wall 97.1%                   …
Exit 84: 0.77% (vs 0.88%) wall 0.053%            Overflow users with exit not 84
Efficiency: 77.2% (vs 74.9%)                           5 uscms1525
Eff   >80%: 50.1% (vs 45.3%)                          71 uscms1832
                                                       …
Only UCSD+Nebraska+Wisconsin+Purdue              Normal users with exit 84
                                                      96 uscms1525
Overflow: 829 (15.7% wall 90.0%) Normal: 5264          2 uscms1832
Exit  0: 95.4% (vs 93.4%) wall 97.3%                   …
Exit 84: 0.72% (vs 0%) wall 0.053%               Normal users with exit not 84
Efficiency: 77.3% (vs 74.2%)                           2 cmspa0029
Eff   >80%: 49.2% (vs 33.8%)                         293 uscms1525
                                                       …
```

Example daily email

(See also poster #233 by M. Tadel)

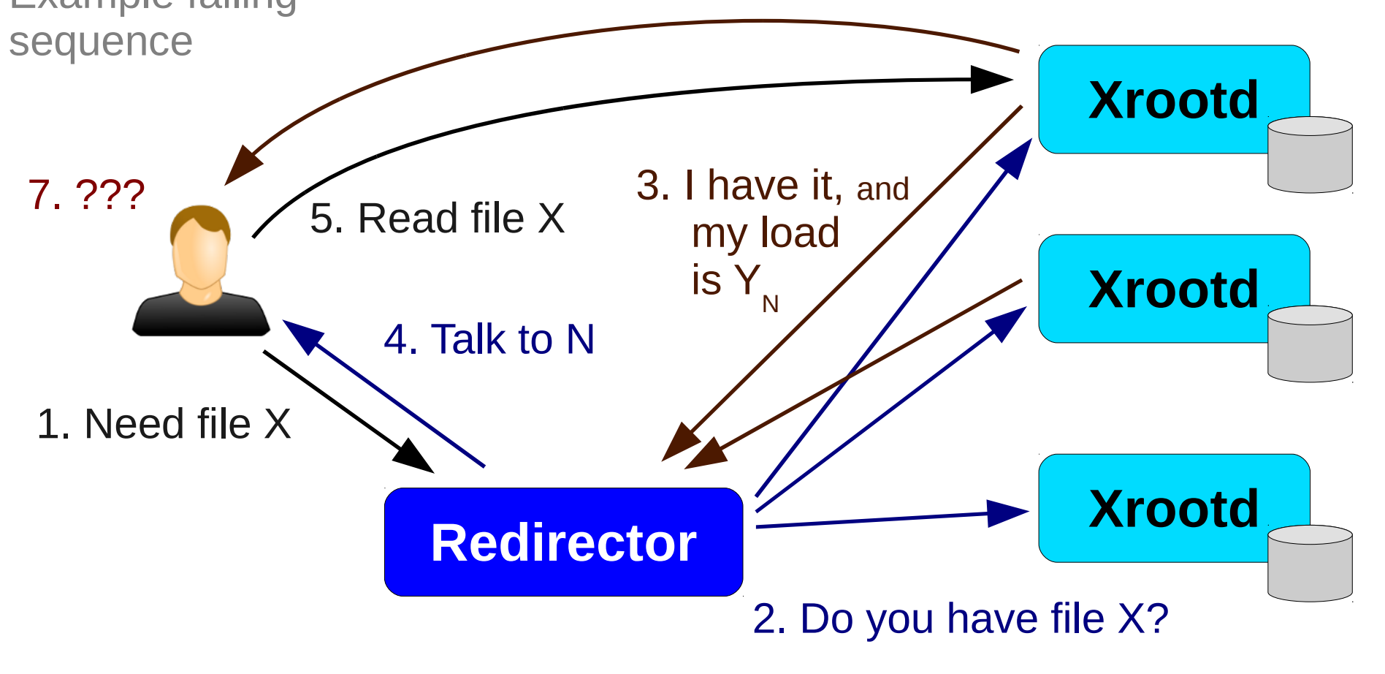
A few problems found:

CMS jobs expect **a site-local configuration file** to point them to the global Xrootd redirector. Overflow glideins must **only run on worker nodes with this setup put in place**, or jobs will fail. We had **added a check** for the appropriate setup at glidein startup. This prevents misconfigured worker nodes from being considered for matching overflow jobs.
However, a significant number of user jobs did fail before we properly diagnosed the problem, showing the importance of proper monitoring and alarms.

If a site-specific service misbehaves, all jobs overflowing to that site will fail.
While not a new problem, the increased scale of the problem prompted us to spend significantly more time writing validation tests for the glideinWMS system, thus drastically reducing the related error rates for user jobs running on both traditional and overflow setup.

Discovered one unexpected flaw in the implementation of the federated Xrootd setup that we used in Fall 2011. The used client **had no fault tolerance**.
When a user contacted the Xrootd redirector with the request of a file, the redirector would query the known Xrootd data-hosting servers, and then point the user to the server which claimed to have the file and be the less loaded. However, if the chosen Xrootd server did not deliver the file, e.g. because the user could not be authenticated or authorized, the user job would fail. If such a misbehaving Xrootd server were to claim to also be very lightly loaded, a large fraction of the overflow jobs could fail due to most requests being redirected to this server.

Example failing sequence



Fixed in 2012 CMS software versions, now will **retry a different server** if the first given fails.
(Still in the process of migrating all CMS users to the fixed CMS SW versions)

This work is partially sponsored by the US National Science Foundation under Grants No. PHY-1104549, PHY-1104447, PHY-1104664 (AAA) and PHY-0612805 (CMS Maintenance & Operations), and the US Department of Energy under Grant No. DE-FC02-06ER41436 subcontract No. 647F290 (OSG).