EXPLOITING VOLATILE OPPORTUNISTIC COMPUTING RESOURCES WITH LOBSTER

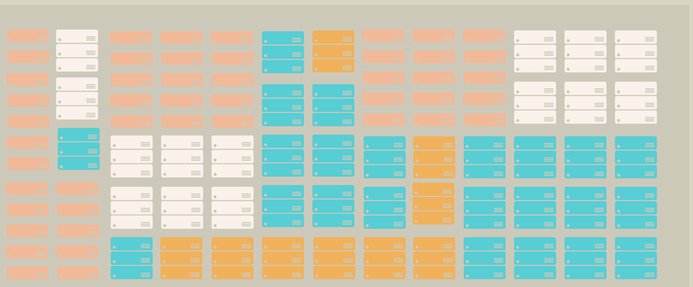
Anna Woodard, Matthias Wolf, Charles Nicholas Mueller, Ben Tovar, Patrick Donnelly, Kenyi Hurtado Anampa, Paul Brenner, Kevin Lannon, Michael Hildreth, Douglas Thain

Lobster is a workflow manager for harnessing non-dedicated resources for high-throughput workloads. Lobster brings together several tools from the Cooperative Computing Lab— Parrot, Work Queue, and Chirp— along with new capabilities to yield a comprehensive system. Because it only requires standard permissions, Lobster can deploy any resource on which the user can run. Lobster features a master-worker architecture, with jobs created on-the-fly, allowing for dynamic job sizing.

WHY LOBSTER?

We have developed Lobster to harness computing resources that are not integrated into the Worldwide LHC Computing Grid (WLCG), including private university clusters, commercial clouds, and other production grids. These systems are not immediately prepared to service High Energy Physics (HEP) workloads, because the required software stacks, data sharing services, and workload management software are not present. Further, they are not dedicated and commonly evict users without warning. Lobster is designed to handle these factors in multiple dimensions: job construction, software delivery, data access, output management, and system monitoring.

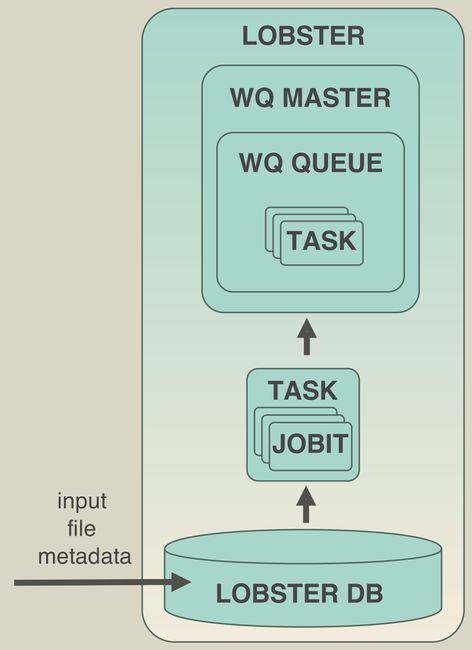
The Notre Dame (ND) Tier 3 (T3) has about 1,000 CPU cores, a homogenous hardware and dedicated software setup, and dedicated support.



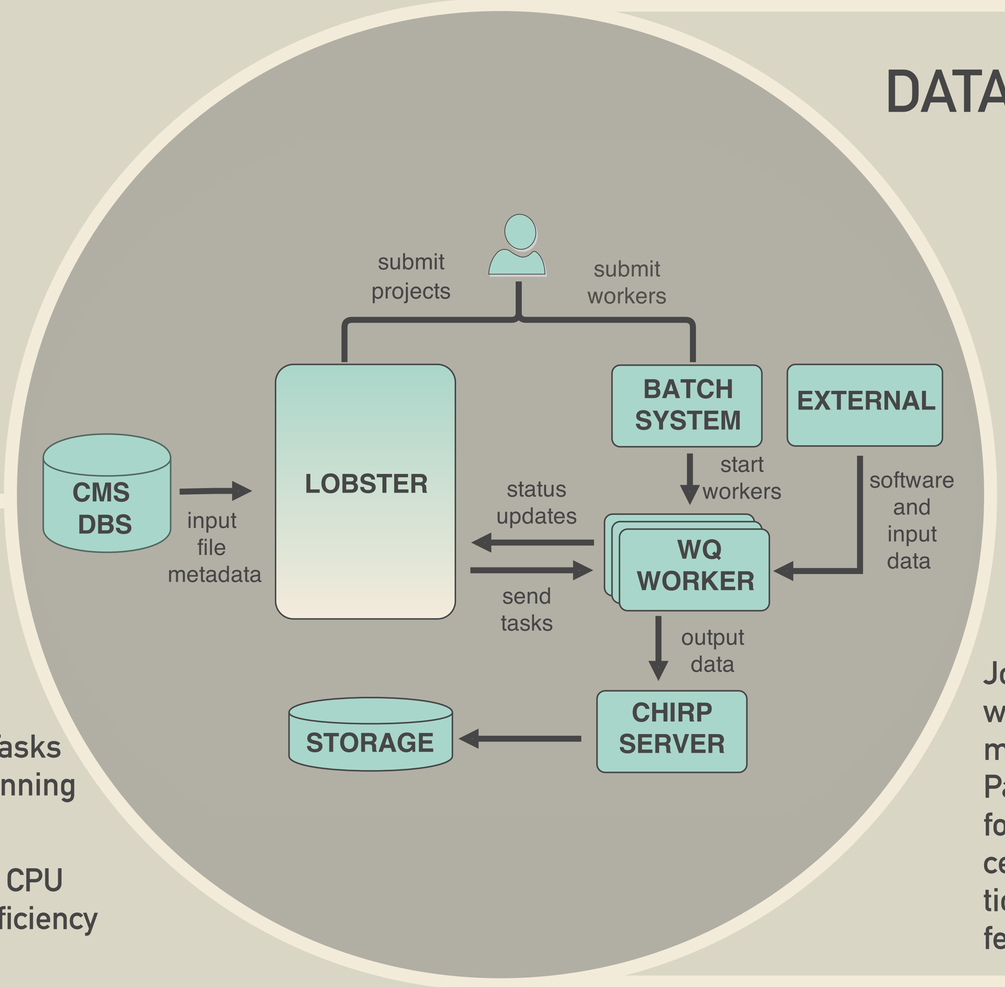
The ND Center for Research Computing (CRC) houses about 21,000 CPU cores, heterogeneous hardware, and no privileged access, specialized software setup, or dedicated support. These cores are available for opportunistic use when idle, but jobs will be evicted when the owner reclaims their resources.

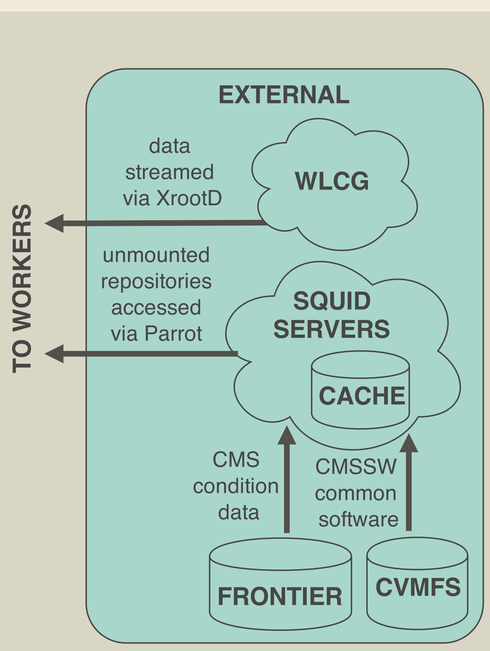
SCHEDULING

Lobster uses the Compact Muon Solenoid (CMS) Database Bookkeeping System (DBS) to obtain file metadata for a given set of datasets to be processed. This is used to construct a database of jobits— in other words, the smallest elements into which the dataset can be divided and still be submitted as a self-contained task to the remote worker. Lobster assembles tasks on-the-fly from an integer number of jobits, which can be adjusted by the user, and schedules them via Work Queue (WQ). The WQ master distributes tasks to workers and reports progress back to Lobster. Optional WQ 'foremen' can be started to reduce the load on the WQ master by mediating between it and the workers.



DATA

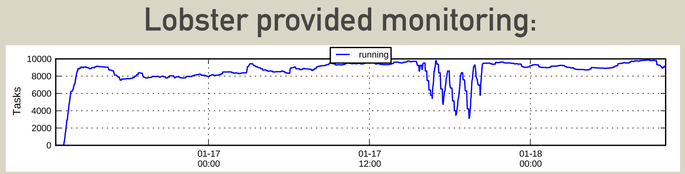




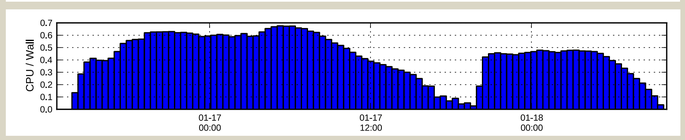
Job scripts and the user sandbox are distributed to the worker via WQ. The CMS software environment is mounted via the CernVM File System (CVMFS) and Parrot. Data is staged via Chirp or streamed via XrootD for local data storage and streamed via XrootD for access to the CMS wide-area data federation. CMS condition data is provided via Frontier. Outputs are transferred via WQ or an optional Chirp server.

MONITORING

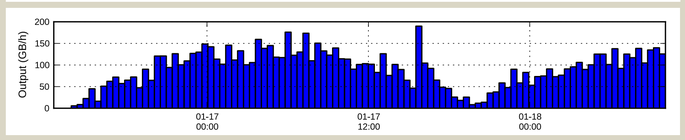
Lobster provided monitoring:



Tasks running



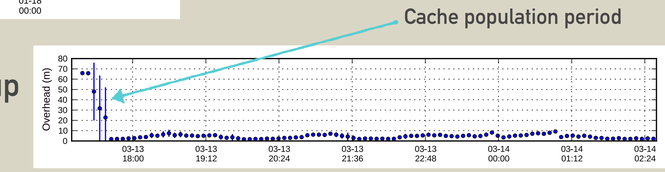
CPU efficiency



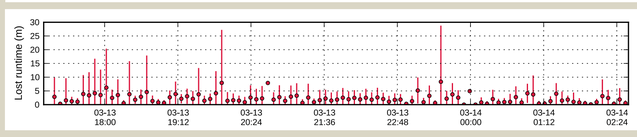
Output transferred

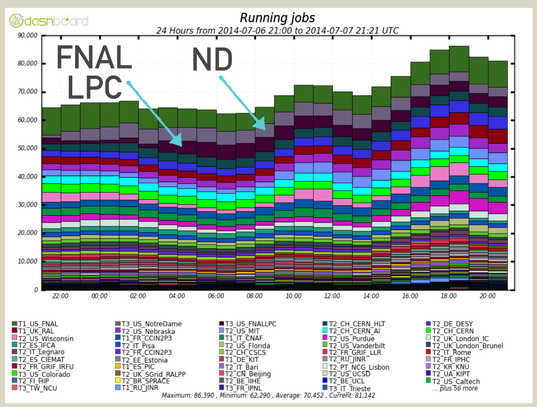
Cache population period

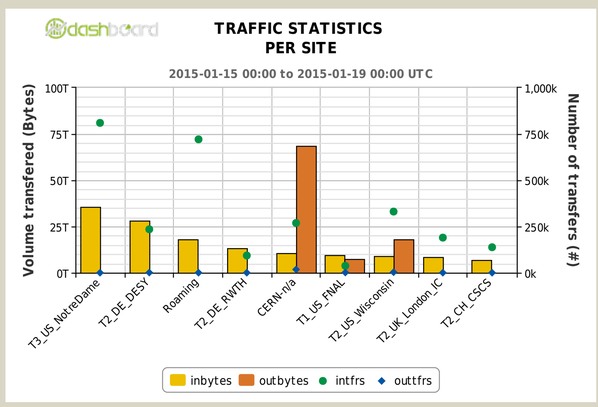
Software setup overhead



Lost job runtime due to eviction and worker connectivity issues

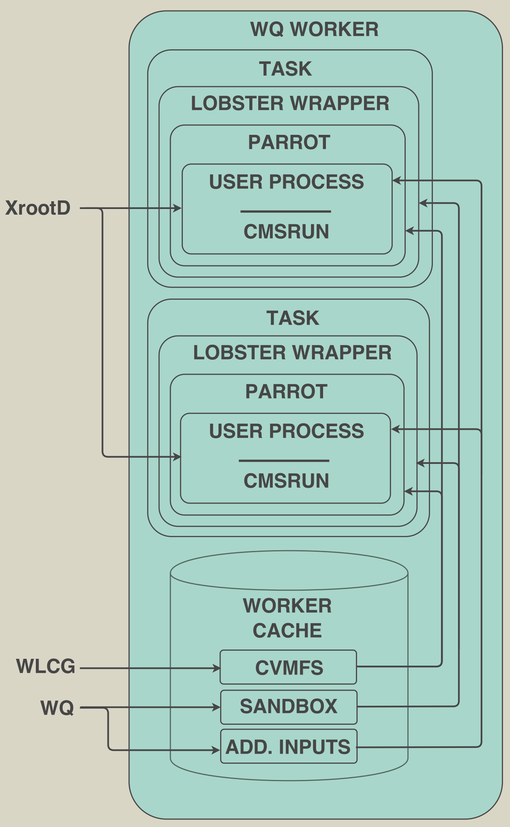






EXECUTION

Workers can be submitted via a variety of batch systems (HTCondor, SGE, PBS, etc.) Each task includes a wrapper which performs pre- and post-processing around the user application. The pre-processing steps checks for basic machine compatibility, obtains the software distribution and optionally stages the input data, and starts the application. The application runs with either FUSE or Parrot to access software at runtime via the CVMFS global file system. The post-processing step sends output data to the data tier and summarizes job statistics, which are sent back to the master. Workers hold resources and run tasks for the master until the work is finished or the worker is evicted. Multicore workers run multiple tasks in parallel, sharing a local cache for CVMFS and WQ files.



We are grateful to many people who assisted in constructing and troubleshooting this complex system; in particular, Jakob Blomer, Dan Bradley, and the CVMFS team; Dave Dykstra, Barry Blumenfeld and the Frontier team; and Serguei Fedorov and the CRC staff.

This work was supported in part by National Science Foundation grant OCI-1148330 and a Department of Education GAANN fellowship.

CMS

UNIVERSITY OF NOTRE DAME