Exploiting volatile opportunistic computing resources with Lobster

Anna Woodard, Matthias Wolf, Charles Mueller, Ben Tovar, Patrick Donnelly, Kenyi Hurtado Anampa, Paul Brenner, Kevin Lannon, Mike Hildreth, Douglas Thain
University of Notre Dame, Notre Dame, IN 46556, USA
E-mail: {awoodard|mwolf3}@nd.edu

Abstract. Analysis of high energy physics experiments using the Compact Muon Solenoid (CMS) at the Large Hadron Collider (LHC) can be limited by availability of computing resources. As a joint effort involving computer scientists and CMS physicists at Notre Dame, we have developed an opportunistic workflow management tool, Lobster, to harvest available cycles from university campus computing pools. Lobster consists of a management server, file server, and worker processes which can be submitted to any available computing resource without requiring root access.

Lobster makes use of the Work Queue system to perform task management, while the CMS specific software environment is provided via CVMFS and Parrot. Data is handled via Chirp and Hadoop for local data storage and XrootD for access to the CMS wide-area data federation. An extensive set of monitoring and diagnostic tools have been developed to facilitate system optimisation. We have tested Lobster using the 20 000-core cluster at Notre Dame, achieving approximately 8-10k tasks running simultaneously, sustaining approximately 9 Gbit/s of input data and 340 Mbit/s of output data.

1. Introduction
Data produced by the experiments at the LHC are generally analyzed on dedicated resources integrated into the Worldwide LHC Computing Grid (WLCG) [1]. Analytical tasks of the high energy physics (HEP) community are complex by nature and require large amounts of data. The sites connected to the WLCG have dedicated system administration teams to ensure the presence of optimal software environments, stable data delivery, and consistent user support. Existing HEP job resource management software is typically intended for use in this specialized environment.

HEP research groups sometimes find that the available WLCG resources are insufficient and may have additional facilities at their disposal that they wish to utilize. These include generic university clusters, commercial cloud computing, and other computing infrastructure systems. Such systems are normally not prepared to run HEP workflows, as they lack the software stack, data connectivity, and infrastructure for common workflow management tools [2, 3, 4] to access them. Privileged access may not be available, and alterations in machine setup may not be permitted. Moreover, on non-dedicated systems, jobs may be evicted without any notice whenever the original owner of the resource reclaims it.
To mitigate these restrictions, we have developed the *Lobster* job management system, which has the goal of making it possible to efficiently perform HEP analysis through optimal utilization of whatever computing resources are available to the user, regardless of whether they are opportunistic or dedicated. We make the user’s life easier by separating task management from management of the resources.

2. Lobster

Lobster contains several components, as shown in Figure 1. All Lobster processes can be run by ordinary users without privileged access. To begin a project, the user starts the Lobster master, which divides the project submitted by the user into a series of tasks. We use Work Queue (WQ) [2], which consists of a library for creating master-worker style programs to manage the queue of tasks and the pool of workers. WQ assigns tasks to workers, notifies the master as tasks are completed, and assigns new tasks to workers as they become available. The user requests resources for the project by executing a script which submits a specified number of workers to the batch system. The user may request resources from one or several clusters, and may request cores which are entirely dedicated, entirely opportunistic, or a combination of both. The workers provide means to access software and external input data, while a user-space Chirp server [9] can accept output files to be stored.

![Figure 1](image_url)

Figure 1. Structural overview of the main components of Lobster. Users start a Lobster master and submit Work Queue workers to the batch system. Both then communicate with each other to run tasks and process data.

2.1. The lobster master

The Lobster master accepts a workflow description provided by the user, which includes the analysis program to run, the dataset to be analyzed, and the destination for the output. Relevant file metadata is downloaded from the Compact Muon Solenoid (CMS) Dataset Bookkeeping System (DBS) to be stored in an internal SQLite database. The project is first split into elements we call jobits, a jobit being the smallest element into which a dataset can be divided and still be submitted as a self-contained task, e.g., a file or a luminosity section. The jobits are combined...
“on the fly” into tasks for assignment to remote workers as they become available. Ideally the task size should be tuned for optimal performance, small enough to minimize computing time losses if a job is evicted, but still sufficiently large to incur only a small penalty through the overhead of runtime setup on the worker node. Currently task size is set by the user, however in the future our intent is for Lobster to determine the optimal task size and adjust it dynamically as conditions change.

The WQ master distributes the tasks to workers and reports progress and job metadata back to Lobster. To ease the load on the WQ master, the user can start additional WQ foremen, which mediate between the master and its workers, caching data and reducing the time the master spends in communication.

2.2. The worker
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 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. Each worker can manage multiple cores concurrently, sharing a single connection to the master as well as a local cache for CVMFS and WQ files.

2.3. Input and output
Lobster receives inputs from several external sources and has several complementary methods for collecting output. An example configuration is shown in Figure 3.

Common parts of the CMS software environment are accessed via the CernVM File System (CVMFS) [3]. If CVMFS is already mounted on the worker node, which is normally the case for workers running on dedicated resources, it is accessed via FUSE. Otherwise, the Parrot virtual filesystem [8] is used. CMS condition data is provided via Frontier [4]. To mitigate high loads on the central CVMFS and Frontier servers, all data is cached via squid servers.

The user can provide a list of input data files directly to the master process or specify a dataset in the CMS Dataset Bookkeeping System (DBS). In the latter case Lobster obtains the file metadata from the DBS. Input data stored on the WLCG is streamed to the workers via XrootD, which can also be used to stream input data that is stored locally. Output files can be transferred from the workers to the Lobster master by WQ.

Optionally, the user can start a local Chirp server which can provide access to both locally available input data and storage of output files directly to the worker. For a large number of jobs and sizeable output files, starting a Chirp server is highly recommended, as serial output transfer via WQ may decrease the Lobster master efficiency.

3. Monitoring
Because of the complex interactions between its distributed components, diagnosing performance problems with can be problematic. For example, excessive loads on a squid server providing analysis software to workers can delay task setups, resulting in excessive task run times and increased eviction rates. To help detect and resolve such issues, Lobster provides the user with a web-based display which provides a variety of performance parameters.

The master process records a variety of performance data during each run, including the status of tasks (created, queued, running, completed, or failed) and the generation of output as files are created and stored. The WQ master process records the creation and eviction of workers, input and output transfer times, and total run time for each task. On the worker nodes,
Metadata for data to be processed is stored in an internal database, and tasks are created on the fly to keep the Work Queue queue filled at a certain level.

Figure 2. Structure of the Lobster master. Metadata for data to be processed is stored in an internal database, and tasks are created on the fly to keep the Work Queue queue filled at a certain level.

Figure 3. Data sources for task execution. Software setup and condition data are read from Frontier and CVMFS, cached via local squid servers. External input data is streamed via XrootD.

Figure 4. Overview of the task execution infrastructure on a worker. Every Work Queue worker can run several tasks in parallel, which execute a wrapper, responsible for basic setup and executing Parrot. Within Parrot, either a user specified script or cmsRun is executed.

the WQ wrapper for each task records the completion of each stage in the execution process, including computing environment setup, task completion, and output transfer.

These records are stored in the Lobster master database and displayed on the monitoring web page. Typical performance plots are shown in Figure 5 and Figure 6. Figure 5 displays task overhead time over the course of a typical project, demonstrating the relatively high overhead early in the run as workers are set up and worker caches are initially populated with experiment data and software transferred from CVMFS. Subsequent tasks on the same worker node begin...
Figure 5. Job overhead as a function of time. For every time segment, the average of the overhead of jobs finishing within that segment is displayed.

Figure 6. Approximately 10,000 running jobs on the opportunistic pool at the University of Notre Dame. The upper panel shows the number of tasks running as a function of time, while the center panel depicts the efficiency of the Lobster system, where the CPU time of all jobs is compared to the overall task run time per time slice. The lower panel displays the transfer rate to the storage element, achieving at least 200 Mbit/s for a longer time period.

with a hot cache and thus have a substantially lower overhead, comparable to that achieved when running on dedicated resources. This is significant because high eviction rates at any point in the project run will also require the creation of new workers, increasing overhead as caches have to be repopulated.

Figure 6 displays a typical project running on about 10,000 total cores. The upper panel shows the number of tasks running concurrently, while the center panel gives the fraction of total CPU time spent in actual processing, providing an estimate of efficiency over the course of the run. Efficiency is initially low due to the overhead of cache population but reaches a sustained peak after several hours. About halfway through the run efficiency begins to decline; this was traced to a transient problem with XrootD which caused massive task failures. The
Figure 7. Excerpt of data accumulated by the CMS Dashboard project. Shown are running tasks on all CMS sites per time bin. The numbers depicted for the University of Notre Dame include opportunistic resources, as only around 1000 cores are available for the dedicated Tier 3 (T3).

master was not able to distribute new tasks fast enough to keep up with demand, which led to some workers running fewer tasks than possible. When the problem is finally resolved, efficiency returns immediately to a high level, since the worker caches are still filled.

Lobster also has support for monitoring through CMS Dashboard. The master and worker processes send status updates to a central CMS Dashboard server, which allows users to compare projects running opportunistically with others running on dedicated resources. An example of this is shown in Figure 7. The Tier 3 computing cluster at the University of Notre Dame, with only about 1000 dedicated cores, is able to run a workload comparable to much larger Tier 1 and Tier 2 sites by using opportunistic resources.

4. Conclusion and Outlook
We have used Lobster successfully to accelerate our HEP workflows at Notre Dame, using up to 10000 cores in addition to the dedicated 1000 cores. This provides a significant improvement to our resource and time usage. For Monte Carlo simulations, use of Lobster has allowed us to shift much of the workload away from dedicated resources.

Chirp requires the user to start a separate server, so we are transitioning to use of Storage
Resource Management (SRM) for stageout. Most of our development effort has been focused on running at Notre Dame. We are working to generalize the software, so that users can more easily run at other sites throughout the HEP community.

Acknowledgments
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.

References
[1] Worldwide LHC Computing Grid, http://wlcg.web.cern.ch/.
[2] M. Albrecht, D. Rajan, and D. Thain. Making Work Queue Cluster-Friendly for Data Intensive Scientific Applications . In IEEE International Conference on Cluster Computing, 2013.
[3] J. Blomer, C. Aguado-Sánchez, P. Buncic, and A. Harutyunyan. Distributing LHC application software and conditions databases using the CernVM file system. Journal of Physics: Conference Series, 331(4):042003, Dec. 2011.
[4] B. Blumenfeld, D. Dykstra, L. Lueking, and E. Wicklund. CMS conditions data access using FroNTier. Journal of Physics: Conference Series, 119(7):072007, July 2008.
[5] M. Cinquilli, D. Spiga, C. Grandi, J. M. Hernandez, P. Konstantinov, M. Mascheroni, H. Riahi, and E. Vaandering. CRAB3: Establishing a new generation of services for distributed analysis at CMS. Journal of Physics: Conference Series, 396(3):032026, 2012.
[6] E. Fajardo, O. Gutsche, S. Foulkes, J. Linacre, V. Spinoso, A. Lahiff, G. Gomez-Ceballos, M. Klute, and A. Mohapatra. A new era for central processing and production in cms. Journal of Physics: Conference Series, 396(4):042018, 2012.
[7] D. Spiga, S. Lacaprara, W. Bacchi, M. Cinquilli, G. Codispoti, M. Corvo, A. Dorigo, A. Fanfani, F. Fanzago, F. Farina, M. Merlo, O. Gutsche, L. Servoli, and C. Kavka. The CMS remote analysis builder (crab). In S. Aluru, M. Parashar, R. Badrinath, and V. Prasanna, editors, High Performance Computing, HiPC 2007, volume 4873 of Lecture Notes in Computer Science, pages 580–586. Springer Berlin Heidelberg, 2007.
[8] D. Thain and M. Livny. Parrot: An Application Environment for Data-Intensive Computing. Scalable Computing: Practice and Experience, 6(3):9–18, 2005.
[9] D. Thain, C. Moretti, and J. Hemmes. Chirp: A Practical Global Filesystem for Cluster and Grid Computing. Journal of Grid Computing, 7(1):51–72, 2009.