DataLab: Transactional Data-Parallel Computing on an Active Storage Cloud

Brandon Rich
Dept of Computer Science and Engineering
University of Notre Dame
Notre Dame IN 46556
brich@nd.edu

Douglas Thain
Dept of Computer Science and Engineering
University of Notre Dame
Notre Dame IN 46556
dthain@nd.edu

ABSTRACT

Active storage clouds are an attractive platform for executing large data intensive workloads found in many fields of science. However, active storage presents new system management challenges. A large system of fault-prone machines with local persistent state can easily degenerate into a mess of unreferenced data and runaway computations. Our solution to this problem is DataLab, a software framework for running data parallel workloads on active storage clusters. DataLab provides a simple language for expressing workloads, works with legacy application codes, and achieves robustness through the use of distributed transactions. Our prototype implementation scales to 250 nodes on a large biometric image processing workload.

Categories and Subject Descriptors

C.4 [**Performance**]: Fault Tolerance; H.2.4 [**Systems**]: Parallel Databases

General Terms

Reliability, Performance

Keywords

Active Storage, Transactions, Cloud Computing

1. OVERVIEW OF DATALAB

DataLab is a system for executing I/O intensive workloads on an active storage cloud of hundreds to thousands of nodes. Large datasets are partitioned across the nodes, while small computations are dispatched to each node using a high level language. Distributed transactions are employed to make the system robust to a wide variety of failures.

Figure 1 shows the components of a DataLab system. An array of *active storage units* (ASUs) provides scalable storage capacity and processing capability. Each is a commodity machine running a user level server that provides remote data access and remote execution within a common security model. A central *database server* keeps track of the ASUs, data sets, distributed file locations, function definitions, and current and and past jobs. A *client* application drives the system, letting the user define and import sets and functions, start and monitor jobs, and view results.

A user of DataLab operates on *sets*, which are named collections of files. Each element of a set can be processed by a *function*, which is an encapsulated program with one or more inputs and outputs and no hidden side effects. Users may define new functions within DataLab, and then invoke them in several ways. `apply` creates a new set by processing each element with a given function. `select` chooses a subset of an existing set. `compare` matches all elements of two sets against each other.

As a motivating example, we consider a workload used by biometrics researchers at the University of Notre Dame, who regularly work with a standard dataset of 60,000 iris images. The same data is processed many times in a common workflow pattern, employing a number of slightly different algorithms for preprocessing and matching images. These workflows are I/O bound and naturally parallel, so they are well suited for processing on an active storage cloud.

First, the user imports data into DataLab:

```
create set IrisTIFF;
import "/tmp/images/*" into IrisTIFF;
```

Suppose this user prefers to work with images in the BMP format. To do this, the user simply applies `ConvertBMP` to all the images in parallel:

```
apply ConvertToBMP on IrisTIFF into IrisBMP;
```

Next, the images must be *segmented*, a process that isolates the the area of the image containing the actual iris, producing three outputs for every input:

```
apply Segment3B on IrisBMP
into  Geometry,Template,Mask;
```

Now, suppose that some of the images contain a certain artifact that can be identified by the program `ContainsArtifact`. To create a new set with only those images:

```
select ContainsArtifact
from   IrisBMP,Mask
into   ArtifactIrises,ArtifactMask;
```


HPDC'08, June 23–27, 2008, Boston, Massachusetts, USA.

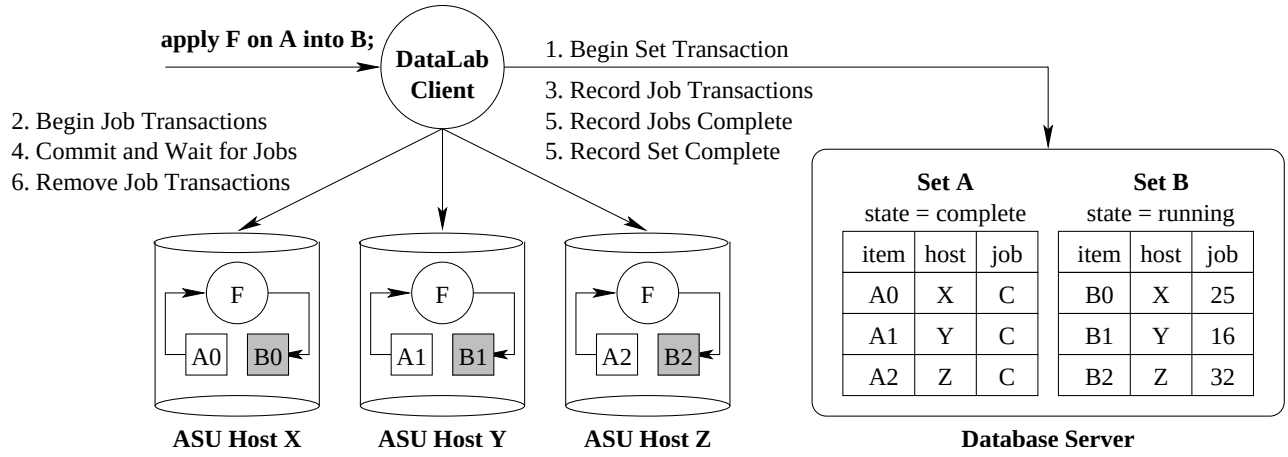

Figure 1: Architecture of DataLab

Finally, the user compares all those images to each other, to observe accuracy in the presence of artifacts:

```
compare Compare2D
on      ArtifactIrises
into    CompareResults;
```

The result of the comparison is simply the concatenated output of each instance of the function Compare2D:

```
iris001.bmp x iris002.bmp = 0.561
iris001.bmp x iris003.bmp = 0.230
...
```

DataLab makes extensive use of *distributed transactions* to permit recovery from failure and avoid runaway computations and unreferenced garbage data. Each ASU exports a *job transaction* interface that provides remote execution using two-phase commit and the ability to query, restart, or abort a single job. Layered on top of this is a *set transaction* interface that coordinates all of the job transactions by recording the necessary details in the shared database server. The set transaction also has two-phase commit, query, restart, and abort capabilities. As a result, the system is highly robust: any component may be disconnected or reset, and the workload will still complete. Likewise, a large workload may be cleanly aborted midstream, even in the face of disconnections.

Figure 2 shows the performance of our DataLab prototype on a network of 250 Linux-based active storage units. The system is processing an iris segmentation workload on 6000 images. The estimated sequential execution time of this workload is fifteen hours. By scaling the system up, the runtime is reduced to less than ten minutes, turning a long batch workload into a nearly-interactive task. While our prototype still has opportunities for further speedups, this demonstrates that the approach is fundamentally sound.

The concept of *active storage* was first proposed by Riedel and Gibson [1] as a restricted computing facility integrated into low-level disk controllers. Later work has focused on applying the active storage concept to existing hardware by treating an entire conventional machine as an *active storage unit* (ASU) [2, 3] within a larger network. Several systems have been proposed for exploting active storage clusters for

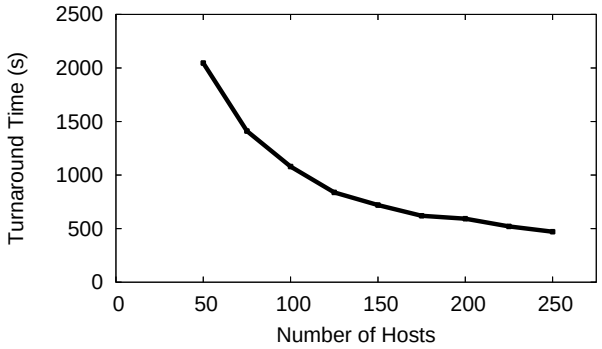

Figure 2: Performance of Prototype

performing *read only* queries on large datasets. For example, MapReduce [4] parallelizes large-scale workloads whose output can be expressed as key-value pairs. Dryad [5] enables the construction of large graphs of pipelined processes that implement complex SQL queries.

The new contribution of DataLab in this area is the use of *distributed transactions* for robustness, a *concrete syntax* for use by non-expert users, and *internal management* of output-intensive operations. Future work will address more complex workflows and scaling to thousands of nodes.

This work was supported by National Science Foundation grants CCF-0621434 and CNS-0643229.

2. REFERENCES

[1] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large scale data mining and multimedia. *VLDB* 1998.

[2] X. Ma and A. Reddy. Implementation and evaluation of an active storage system prototype. *Workshop on Novel Uses of System Area Networks*, 2002.

[3] R. Wickremisinghe, J. Vitter, and J. Chase. Dist. comp. with load managed active storage. *HPDC* 2002.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large cluster. *OSDI* 2004.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data parallel programs from sequential building blocks. *EuroSys* 2007.