UCRL-CONF-209265

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Middleware for Astronomical Data Analysis Pipelines

*Abdulla G., Liu D., Garlick J., Miller M.,
Nikolaev S. Cook K., Brase J.*

January 26, 2005

ADASS XIV, Pasadena, California
October 24-27, 2005

ADASS XIII
ASP Conference Series, Vol. XXX, 2004
F. Ochsenbein, M. Allen and D. Egret eds.

Middleware for Astronomical Data Analysis Pipelines

Ghaleb Abdulla, Jim Garlick, Marcus Miller, Sergei Nikolaev, Kem Cook, Jim Brase

Lawrence Livermore National Laboratory

David Liu

University of California, Berkeley

Abstract. In this paper we describe our approach to research, develop, and evaluate prototype middleware tools and architectures. The developed tools can be used by scientists to compose astronomical data analysis pipelines easily. We use the SuperMacho data pipelines as example applications to test the framework. We describe our experience from scheduling and running these analysis pipelines on massive parallel processing machines. We use MCR, a Linux cluster machine with 1152 nodes and Luster parallel file system as the hardware test-bed to test and enhance the scalability of our tools.

1. Introduction

Large sky surveys present new challenges in data management, such as the increased quantity and pace of data produced by new instruments, the need to process it in near real-time for alerts, and the inclusion of time domain data in the science. These challenges are expected to be met with sophisticated pipelines and database systems running on high performance parallel computers.

Some of the complexity of these software systems can be moved into middleware which provides a data-driven execution framework for pipelines and an abstraction layer for the operating system interfaces of a parallel computer. Data-centric middleware can offer the following benefits:

- Data flow is described using a data description language (DDL) instead of imperative scripts. The DDL is easier to write and maintain, it expresses more information than scripts, and provides mechanisms for formalizing the description of the interfaces of individual algorithms which is good software engineering practice and promotes modularity and reuse.
- With knowledge of pipeline data flow and data dependencies in the system, the middleware can automatically manage the details of data-parallel[1] execution.

[1] In this context, data-parallel refers to parallel execution that is possible due to a lack of dependencies in the data.

- Key aspects of the parallel operating system, for example MPI job launch and monitoring, are abstracted and thus allowed to change over the life of the survey without impacting the entire system.
- Data provenance and memoization services of the middleware can minimize the resources required to recompute science data when some but not all of the pipeline software is incrementally improved.
- When possible, the data is self-describing and not coupled to the algorithms.

We are simultaneously evaluating current pipeline technology and new research in data-centric middleware. In addition we are experimenting with efficient ways to modularize algorithms. In order to understand the challenges, we have initially focused on porting the SuperMACHO data set and pipelines to large scale LLNL platforms such as MCR[2], and have attempted to run it in conjunction with two prototype software packages: the GridDB system from Berkeley (Liu 2003), a grid-oriented tool with many of the features listed above; and Industrial Strength Pipes (ISP) from Livermore (Garlick 2004), a simple pipeline construction tool with parallel execution capability. These two systems are described below.

2. GridDB

As illustrated in Figure 1, GridDB provides a veneer, or overlay, on top of existing process-centric grid services (i.e. globus and condor) that enables clients to create and manage grid computations, as well as interact with their results. GridDB provides a host of benefits, including declarative interfaces, type checking, interactivity, data provenance and memoization, which is the ability to store and retrieve, rather than unconditionally execute, computations. GridDB works on top of process-centric middleware, rather than replacing it. Consequently, users can continue employing their pre-existing imperative data processing codes and programming knowledge.

The GridDB framework is a good starting point for achieving many of our goals: modularity, data-centricity, efficient execution and support for real-time processing. We are leveraging it to test the viability of these concepts in realistic scenarios. As such, we expanded GridDB's implementation by porting it onto MCR. We also specified the SuperMACHO image processing application within the GridDB framework.

While GridDB provides many benefits derived from its data-centricity, it has one major liability: it incurs performance penalties due to its reliance on a database system (GridDB can operate on top of any SQL92-compliant database). Because there are penalties associated with accessing a database system, pipelines executed through GridDB may execute slower than pipelines executed through process-centric middleware.

Our initial experiments with the GridDB prototype (which uses PostgreSQL as its database) indeed exhibits performance problems, for example boundary crossings between GridDB and its database component lengthen the critical

[2]MCR is a 2304-CPU Xeon-based Linux cluster at LLNL.

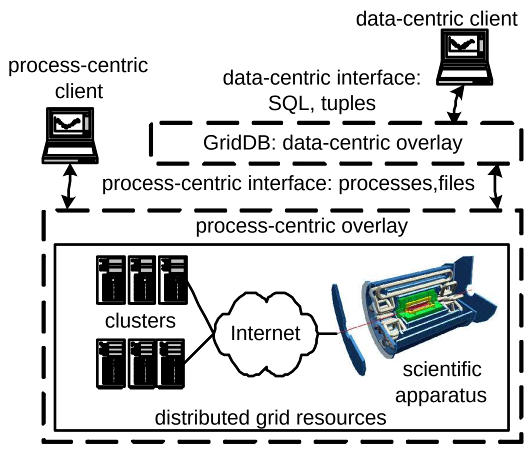
data-centric client
process-centric client
data-centric interface: SQL, tuples
GridDB: data-centric overlay
process-centric interface: processes,files
process-centric overlay
clusters
Internet
scientific apparatus
distributed grid resources

Figure 1. GridDB Architectural Overview

path in pipeline execution. Additionally, our prototype has revealed performance problems in boundary crossings between GridDB and the file system. Currently, we are optimizing such boundary crossings using a series of optimization techniques, include caching, batching and asynchronous processing. We also plan on implementing a framework for prioritization of server tasks during CPU-bound processing.

3. Industrial Strength Pipes

Industrial Strength Pipes (ISP) is a prototype pipeline construction set for parallel systems (it is not a complete middleware solution). It allows filters to be constructed by binding algorithms to ISP library calls. The filters are compiled as separate executables that can be chained together into a pipeline using a UNIX shell in the same manner as regular UNIX pipelines. Instead of carrying unstructured data, the pipes carry XML which contains a stream of *work unit* data structures. Each filter reads a stream of work units on its standard input, and writes a stream of work units to its standard output.

The algorithm in a filter is registered as a callback that runs once per work unit. The work unit contains two types of data: *metadata*, passed by value, and *files*, passed by reference. Multiple files and metadata may be present in a the work unit, indexed by a string key. The algorithm may use ISP calls to read, write, or modify some or all of the files and metadata referenced in the work unit by key. After the algorithm finishes, the work unit is passed downstream for subsequent processing by other filters.

In an image processing application, for example, a filter might turn a collection of raw image files into a stream of work units containing file references. The next filter might, for each work unit, read the raw image and write a calibrated image and some metadata, which are added to the work unit. A subsequent filter might read the calibrated image and metadata and produce a list of features in a file which is added to the work unit. The last filter might add pertinent data products to a database and clean up.

Unless otherwise noted by the algorithm when it registers itself, work units are assumed to be independent of one another and can be processed out of order. On a parallel system, ISP exploits this implicit data-parallelism. As soon as a work unit becomes available as input to a filter, ISP's execution engine submits a request to ISP's parallel scheduler to allocate CPU's to run the algorithm. As CPU's become available, the scheduler executes the requests using existing process-centric middleware in FIFO order, passing the XML work unit to and from the remote process. Because files are passed by reference, this strategy presumes that file references can be followed on any node of a parallel system (a *global file system*), and that a file written and closed on one node can be safely opened and read on another (*close-to-open cache coherency*).

A work unit can contain multiple file references and metadata that are indexed by a string key. While every filter can produce, consume, and read (without consuming) files and metadata by key, filters need not access all of the files and metadata in the work unit. Some can pass through to downstream filters. Data cannot, however, flow upstream, thus the flow of data in the pipeline is a directed, acyclic graph.

We have successfully run data through several of the algorithms of the SuperMACHO pipeline wrapped up as ISP filters on MCR. This test has demonstrated that ISP neatly hides the details of parallel execution on a cluster, and that no scaling issues exist with ISP at least up to the 64 CPU level (we have not tested with more CPU's yet). It has also shown that leveraging the UNIX pipe abstraction has some useful benefits: other UNIX tools can be brought to bear on the XML stream for debugging, and the pipeline can be developed and tested in a serial (e.g. laptop) environment without any of the infrastructure needed to run in parallel present to complicate the process.

4. Conclusions

Our initial implementation of example astronomy pipelines using our proposed middleware is promising. We are in the process of optimizing the GridDB implemenation to provide users with efficient data- centric services. At the same time we are working on adding more data-centric functionality to ISP. ISP or a similar system may be useful as the interface between algorithms and a system like GridDB and we are currently exploring the benefits of the this idea.

ISP's parallel scheduler may open up avenues for research on *pipeline aware schedulers*. We anticipate that there is an opportunity to design scheduling algorithms that can schedule available resources in the "best" way when there are not enough CPU's to run all the pending work.

References

Liu, D., Franklin, M., Parekh, D. 2003, SIGMOD

Dean, J., Ghemawat, S. 2004, OSDI

Garlick, J. 2004, LLNL Presentation, UCRL-PRES-209205