

Hyrax: Demonstrating a New Foundation for Data-Parallel Computation

Vinayak Borkar
University of California, Irvine
vborkar@ics.uci.edu

Michael Carey
University of California, Irvine
mjc Carey@ics.uci.edu

ABSTRACT

We demonstrate Hyrax, a new runtime platform for data-parallel computation under development at UC Irvine under the ASTERIX project. We show the versatility of Hyrax by using it to run XQuery queries from ASTERIX, Hadoop MapReduce jobs using a Hadoop emulation layer, and SQL queries originating from Hive.

1. INTRODUCTION

Writing parallel programs to work correctly and efficiently on a shared nothing cluster of computers is hard. Recently, systems such as MapReduce [5], Hadoop [5], and Dryad [6] have been developed to help programmers solve large data-parallel problems while keeping the programming model simple. In addition, higher level frameworks such as Sawzall [8] on MapReduce, Pig [7], Hive [2], and Jaql [3] on Hadoop, and Dryad/LINQ [10] and Scope [4] on Dryad have arisen to further improve developer productivity. We demonstrate Hyrax as an alternative to MapReduce, Hadoop, and Dryad for solving data-parallel problems. MapReduce and Hadoop provide developers with a simple programming model at the cost of expressiveness of tasks, while Dryad provides a rich set of primitives for expressing arbitrary acyclic data flows, making it a very low level of abstraction for implementing data-parallel computation. Hyrax balances the need for expressiveness while providing out-of-the-box support for commonly occurring communication patterns and operators in data-oriented tasks.

At UC Irvine, ASTERIX [1] is a new project about ingesting, storing, managing, indexing, querying, analyzing, and subscribing to vast quantities of semi-structured data. Hyrax is built with the aim of providing an efficient abstraction for implementing the runtime for ASTERIX.

2. HYRAX ARCHITECTURE

Hyrax runs on a shared-nothing cluster of commodity computers with local CPUs, memory, and disks. Figure 1 shows the high level architecture of Hyrax. Some nodes are

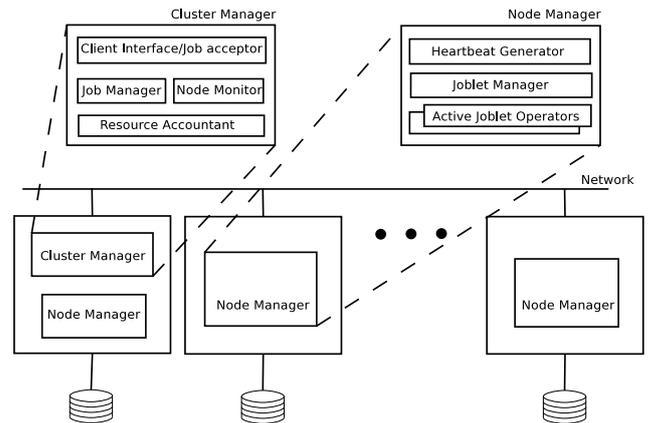


Figure 1: Hyrax system architecture

designated as “cluster managers” and assume the responsibility of coordinating and monitoring “node managers”. A cluster manager is responsible for accepting job creation requests from clients, breaking a job up into discrete stages, deciding parallelization and placement of operators in each stage on the cluster, and initiating job distribution to the node managers. In addition to job management, a cluster manager also receives periodic heartbeats and resource utilization metrics from node managers which it uses in job planning and restarting failed stages. Node managers are responsible for accepting job fragments from a cluster manager and executing them using local resources. The part of a job that is deployed on a node manager is called a joblet. A joblet holds a collection of operators that perform computation on that node. Operators may exchange data with other operators on the same node (using in-process communication) or with operators on other nodes (using network communication). Whenever in-process communication can be used, Hyrax does so without the overhead of data serialization and deserialization. Periodically, node managers send heartbeat messages to cluster managers to show liveness and to relay resource utilization on the node (e.g., CPU, memory and disk activity).

3. HYRAX JOBS

Clients specify Hyrax jobs as Hyrax Operator Descriptor nodes (HODs) connected to each other by Hyrax Connector Descriptors (HCDs) to form an acyclic directed graph. A HOD node is similar to an execution plan tree node as

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD

Copyright 2010 ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

described in [9]. On receiving such a job specification, the cluster manager first replaces every HOD with one or more Hyrax Activity Nodes (HAN). The process of creating a set of HANs for a HOD is delegated by Hyrax to the HOD object. This allows for an extensible operator framework.

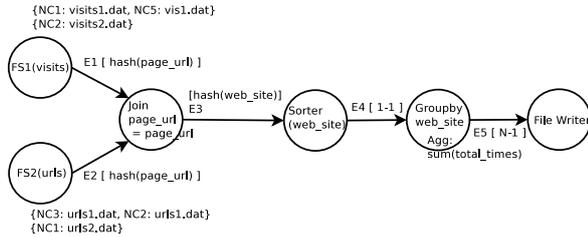


Figure 2: Example Hyrax job specification

Figure 2 shows an example specification of a job that computes the total time spent by visitors on a web site. The job gets its inputs from two logical sources called “visits” and “urls”. Let us assume that “visits” contains (userid, page_url, time_spent) triplets and “urls” contains (page_url, web_site) pairs. To compute the total time spent by visitors on a web-site, the two inputs are joined on the “page_url” fields. The output of the join is grouped on the “web_site” field, and finally the “total_time” field is summed up for each group. FS1 and FS2 in the figure are file scanner HODs that are parameterized with the location information of their inputs. As shown, FS1 gets its input in the form of two partitions. The first partition is replicated on the Hyrax node called NC1 in a file called visits1.dat and on NC5 in a file called vis1.dat. The second partition is available only at NC2 in a file called visits2.dat. Similarly, FS2 is configured with an input that is partitioned two ways. The two scanners feed their outputs to a Join HOD. The Join HOD in this example uses the hash-join algorithm. A sorter HOD is used to sort the join output on the web_site field so that the grouping HOD can operate in constant space. Finally, the File Writer HOD is used to produce the result.

Edges in a job specification are annotated with the type of data distribution to use. For example, E1 and E2 use a hash-based distribution on page_url to distribute the file data to the join instances, routing data from the two sides that match on the page_url field to the same join operator instance. Similarly, a hash distribution is used by E3 to partition data among the sorters on the web_site field. E4 is a 1-1 edge that pipes the output of one producer to one consumer. Finally, E5 is an N-1 edge that merges the outputs of the aggregators into the file writer instance.

Figure 3 shows the result of expansion of the HOD graph into the HAN graph. Notice that the original join HOD has been replaced with two activity nodes – one to perform the build phase of the hash table for the hash-join, and the other to perform the probe on the hash table. A dotted arrow from the builder to the probe indicates a constraint that the probe cannot begin until the builder has completed. Hyrax guarantees that all activity nodes derived from the same HOD are co-located on the same Hyrax node at runtime so they can share information (e.g., the hash table for the join and the run files for the sorter). Note that the two inputs of the original join HOD are now connected to the HANs that actually use them. Similarly, the sort HOD has been replaced with run generation and merging activity nodes.

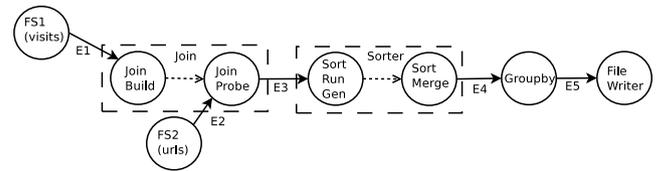


Figure 3: Example Hyrax Activity Node graph

The next step in job planning is to split it into stages. All HANs that are connected only by non-blocking edges are grouped into a stage, splitting the complete job into independent stages. The stages are planned and executed by Hyrax in such a way that all of a stage’s dependencies are complete before it is executed. Hyrax plans each stage in order to decide how many instances to create (and where) for each HAN in the stage. To aid this decision, Hyrax asks HANs for any node affinities (e.g., the file scanner must indicate that it should be instantiated where some replica of each partition of the file can be accessed). The job specification can also include constraints regarding the degree of partitioned parallelism of each operator. In addition, Hyrax provides the stage planner with the ability to gather statistics (if any) from preceding stages and use them for planning. Once each stage is planned, the HAN instances are instantiated in participating Hyrax node managers to form a DAG of Hyrax Operator Nodes (HONs) at runtime. Note that the step of planning the degree of parallelism and placement of HONs is delayed until the stage is ready to run – this allows current cluster conditions (maintained by the Resource Accountant) to play a role in the process. The HONs are responsible for consuming data from their inputs, performing computation, and sending outputs to their consuming HONs. Figure 4 shows the HON graph for the example. The dotted polygons bound the HONs that form a stage.

4. DATA SUPPORT IN HYRAX

Hyrax provides support for expressing data type specific operations. Data between operators flows over connectors in the form of records that contain an arbitrary but fixed number of fields. The type of each field is described by the developer by providing an implementation of a descriptor interface that allows Hyrax to perform common tasks such as serialization, deserialization, comparison, and hashing when necessary. The Hyrax library contains type descriptors for most basic types (e.g., numbers, text, etc). A record as the carrier of data is a generalization of the (key, value) concept in MapReduce and Hadoop. The advantage of this generalization is that operators do not have to artificially package multiple data objects into a single “key” or “value” object. The use of multiple fields for sorting or hashing is natural in this model. For example, as part of the Hyrax operator library, there is an external sort HOD that can be parameterized with the fields to use for sorting. The operator uses type descriptors specified for its input fields in the job description for comparisons and also for marshaling and unmarshaling during spilling. These type descriptors are similar to the Writable and WritableComparator interfaces in Hadoop. However, a subtle difference is that Hyrax does not require that the object instances that flow between the operators implement a specific interface. This allows Hyrax to move data produced and consumed by systems that have

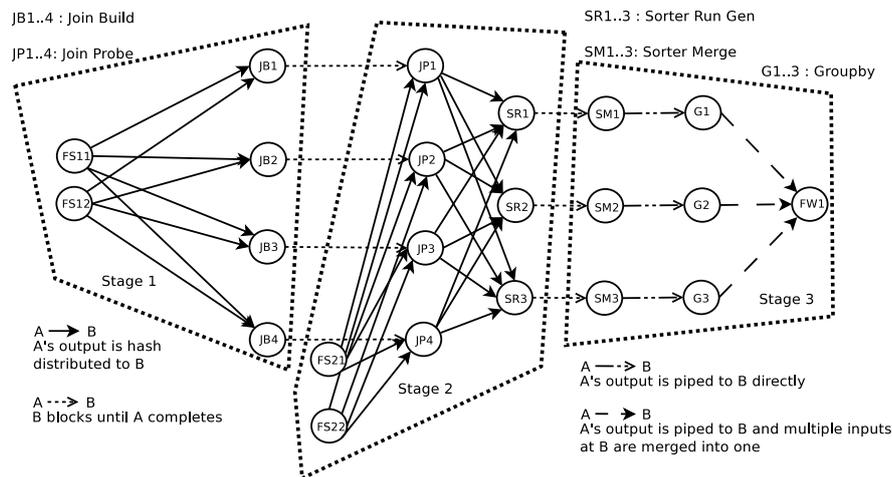


Figure 4: Parallel instantiation of the example

no knowledge of the Hyrax platform.

5. DEMO SCENARIOS

The following scenarios demonstrate the capabilities of Hyrax as a substrate for implementing data-parallel computation.

ASTERIX XQuery on Hyrax

ASTERIX compiles XQuery queries against partitioned XML collections stored at various nodes in the cluster into Hyrax jobs. The aim of this portion of the demo is to show the use of Hyrax as an efficient platform for semi-structured data management.

```
for $v in collection('visits')
for $u in collection('urls')
where $v/page_url eq $u/page_url
group $u as $p by $u/web_site as $ws
return
<total ws="{ $ws }" tot="{ sum($p/total_time) }"/>
```

The XQuery above generates a Hyrax job similar to the example in figure 2. The demo audience will be invited to suggest queries to be executed and will also be shown the resulting HAN and HON graphs.

Hadoop Emulation on Hyrax

The Hyrax operator library contains operators that can be parameterized with implementations of the Hadoop Mapper and Reducer interfaces. In this part of the demo, Hyrax natively executes MapReduce jobs implemented for use with Hadoop. We show the generality of Hyrax by accepting a DAG of Hadoop jobs and exploiting opportunities for parallelism within and across them. The audience will be invited to choose from among several illustrative examples.

Hive on Hyrax

Hive [2] is a SQL processor built to execute as MapReduce jobs on top of Hadoop. In this demo we explore two ways of executing Hive on top of Hyrax – the MapReduce jobs created by Hive can be executed using the Hadoop emulation layer in Hyrax, or Hive can be modified to produce native

Hyrax jobs making the specification more natural and the execution more efficient. The audience will be shown both ways of executing SQL queries on Hive.

6. REFERENCES

- [1] ASTERIX Website. <http://asterix.ics.uci.edu/>.
- [2] Hive Website. <http://hadoop.apache.org/hive>.
- [3] Jaql Website. <http://jaql.org/>.
- [4] R. Chaiken, B. Jenkins, P. A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2):1265–1276, 2008.
- [5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI '04*, pages 137–150, December 2004.
- [6] M. Isard, M. Budi, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA, 2007. ACM.
- [7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, New York, NY, USA, 2008. ACM.
- [8] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall.
- [9] M. G. University, M. N. Garofalakis, and Y. E. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *In Proceedings of the 23rd VLDB Conference*, pages 296–305, 1997.
- [10] Y. Yu, M. Isard, D. Fetterly, M. Budi, A. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In R. Draves and R. van Renesse, editors, *OSDI*, pages 1–14. USENIX Association, 2008.