ASTERIX: An Open Source System for “Big Data” Management and Analysis (Demo)

Sattam Alsubaiee, Yasser Altwowm, Hotham Altwajry, Alexander Behm, Vinayak Borkar, Yingyi Bu, Michael Carey, Raman Grover, Zachary Heilbron, Young-Seok Kim, Chen Li, Nicola Onose; Pouria Pirzadeh, Rares Vernica; Jian Wen

Information Systems Group, University of California, Irvine
mjcarey@ics.uci.edu

ABSTRACT

At UC Irvine, we are building a next generation parallel database system, called ASTERIX, as our approach to addressing today’s “Big Data” management challenges. ASTERIX aims to combine time-tested principles from parallel database systems with those of the Web-scale computing community, such as fault tolerance for long running jobs. In this demo, we present a whirlwind tour of ASTERIX, highlighting a few of its key features. We will demonstrate examples of our data definition language to model semi-structured data, and examples of interesting queries using our declarative query language. In particular, we will show the capabilities of ASTERIX for answering geo-spatial queries and fuzzy queries, as well as ASTERIX’ data feed construct for continuously ingesting data.

1. INTRODUCTION

We started the ASTERIX project [1, 3] at UC Irvine approximately two and a half years ago. Our goal at the outset was to design and implement a highly scalable platform for information storage, search, and analytics. By combining and extending ideas from semistructured data management, parallel database systems, and first-generation data-intensive computing platforms (MapReduce and Hadoop), ASTERIX was envisioned to be a parallel, semistructured information management system with the ability to ingest, store, index, query, analyze, and publish very large quantities of semistructured data. ASTERIX is well-suited to handle use cases ranging all the way from rigid, relation-like data collections, whose types are well understood and invariant, to flexible and more complex data, where little is known a priori and the instances in data collections are highly variant and self-describing.

While ASTERIX began with the objective of creating a parallel, semistructured information management system, three reusable software layers have resulted from our work. The bottommost layer of the ASTERIX stack is a data-intensive runtime called Hyracks [4].

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. Articles from this volume were invited to present their results at The 38th International Conference on Very Large Data Bases, August 27th - 31st 2012, Istanbul, Turkey. Proceedings of the VLDB Endowment, Vol. 5, No. 12
Copyright 2012 VLDB Endowment 2150-8097/12/08... $ 10.00.

Hyracks sits at roughly the same level of the architecture that MapReduce (Hadoop) does in implementations of higher-level data analysis languages such as Pig [8], Hive [2] or Jaql [5]. The topmost layer of the ASTERIX stack is a parallel DBMS, with a full, flexible data model (ADM) and a query language (AQL) for describing, querying, and analyzing data. AQL is comparable to languages such as Pig, Hive, or Jaql, but ADM and AQL support both native storage and indexing of data as well as access to external data residing in a distributed file system (e.g., HDFS). In between these layers sits Algebricks, a model-agnostic, algebraic "virtual machine" for parallel query processing and optimization. Algebricks is the target for AQL query compilation, but it can also be the target for other declarative data languages. Figure 1 summarizes the layers. Sections 2, 3 and 4 briefly explain how each layer does its job, and what its potential value is as a software resource for the “Big Data” analytics and management community. We will start at the bottom of the stack and work our way up to ASTERIX proper (i.e., to the complete end-user system).

2. THE HYRACKS RUNTIME

The Hyracks layer of ASTERIX is the bottom-most layer of the stack. Hyracks is the runtime layer (a.k.a. executor) whose job is to accept and manage data-parallel computations requested either by direct end users of Hyracks or (more likely) by layers above it in the ASTERIX software stack.

Jobs are submitted to Hyracks in the form of directed acyclic graphs that are made up of “Operators” and “Connectors”. To illustrate the key ideas in Hyracks, Figure 2 shows an example Hyracks job representing a TPCH-like query that performs a join between a partitioned file containing Customer records and another parti-
tioned file containing Order records. The result of the join is aggregated to count the popularities of orders by market segment. Nodes in the figure represent Operators, and edges represent Connectors. In Hyracks, Operators are responsible for consuming partitions of their inputs and producing output partitions. Connectors perform redistribution of data between different partitions of the same logical dataset. For example, in Figure 2, the file scanners that scan the Customer and Order files are each connected to the Join Operator by means of an M:N Hash Partitioning Connector. This Connector ensures that all Customer (Order) records reaching the Join Operator partitions agree on the hash-value of the Customer’s (Order’s) CID attribute, thereby enabling each partition of the Join Operator to perform a local join to produce output partitions. In order to perform aggregation on the M:K SEGMENT attribute, the Join Operator’s output partitions are redistributed to the partitions of the GroupBy Operator using another M:N Hash Partitioning Connector; this one hashes on the M:K SEGMENT attribute to ensure that all records that match on the grouping attribute are directed to the same grouping partition. Finally, the GroupBy Operator’s output is written to a file by the FileWriter Operator. The use of a 1:1 Connector between the GroupBy Operator and the FileWriter Operator results in the creation of as many result partition files as GroupBy Operator partitions.

Hadoop has quickly become the gold standard in the industry for a highly scalable data-intensive MapReduce platform. Any system that hopes to displace it must provide a low-cost migration path for existing Hadoop artifacts. To that end, we have built an adapter on top of Hyracks that accepts and executes Hadoop MapReduce jobs without requiring code changes from the user. More details about Hyracks’ computational model, its implementation and performance as well as the Hadoop compatibility layer are available in [4].

3. THE ALGEBRICKS ALGEBRA LAYER

Algebricks is a model-agnostic, algebraic layer for parallel query processing and optimization. This layer’s origin was as the center of the AQL compiler and optimizer of the ASTERIX system. To be useful for implementing arbitrary languages, Algebricks has been carefully designed to be agnostic of the model of its processed data. Logically, operators work on collections of tuples containing data values. The data values carried inside a tuple are not specified by the Algebricks toolkit; the language implementor is free to define any value types as abstract data types. For example, a user implementing a SQL compiler on top of Algebricks would define SQL’s scalar data types to be the data model and would implement interfaces to perform operations such as comparison and hashing. ASTERIX has a richer set of data types, and these have been implemented on top of the Algebricks API as well.

Algebricks consists of the following parts:
1. A set of logical operators,
2. A set of physical operators,
3. A rewrite rule framework,
create type TweetMessageType as open {
  tweet_id: string,
  user:{
    screen-name: string,
    lang: string,
    friends_count: int32,
    statuses_count: int32,
    name: string,
    followers_count: int32
  },
  sender-location: _point ?
  send-time: datetime,
  referred_topics: {{string}}
  message-text: string
};

create dataset TweetMessages(TweetMessageType)
  partitioned by key tweet_id;

(a) ADM type and dataset for tweets

for $tweet in dataset('TweetMessages')
  where some $topic in $tweet.referred-topics
     satisfies contains ($topic, 'verizon')
  for $topic in $tweet.referred-topics
  group by $topic with $tweet
  return {
    "topic": $topic,
    "count": count($tweet)
  }

(b) An example query over Tweet messages to filter
and aggregate tweets

[ {
  "topic": "verizon", "count": 3
},
  { "topic": "commercials", "count": 1
},
  { "topic": "att", "count": 1
},
  { "topic": "at&t", "count": 1
}
]

(c) Result of example query on example data.

(d) Example tweet data

Figure 4: Example AQL schemas, queries, and results

where little is known a priori and the instances in data collections
are highly variant and self-describing. The ASTERIX data model
(ADM) is based on borrowing the data concepts from JSON [6]
and adding additional primitive types as well as type constructors
borrowed from object databases [7]. Figure 4(a) illustrates ADM
by showing how it could be used to define a record type for model-
ing Twitter messages. The record type shown there is an open type,
meaning that the instances of this type will conform to its specifica-
tion but are allowed to contain arbitrary additional fields that vary
from one instance to the next. The example also illustrates how
ADM includes features such as optional fields with known types
(sender-location), nested collections of primitive values (referred-
topics), and nested records (user). More information about ADM
can be found in [3].

Figure 4(d) shows an example of a set of TweetMessageType
instances. Data storage in ASTERIX is based on the concept of a
5. DEMONSTRATION DETAILS

We will demonstrate the following features of ASTERIX.

1. Asterix Data Model (ADM): We will demonstrate how ADM is well-suited to handling use cases ranging from rigid, relation-like data collections, whose types are well understood and invariant, to flexible and potentially more complex data where little is known a priori and the instances in data collections are highly variant and self-describing.

2. Asterix Query Language (AQL): We will demonstrate the capabilities and expressive power of AQL by running different kinds of queries against a running instance of the ASTERIX system. For each of the queries, we will give a walkthrough tour of the system, describing how a logical plan for a query if formed, translated and optimized into an efficient parallel physical plan by the underlying language-neutral Algebrics layer and eventually executed as a DAG by the Asterix runtime - Hyracks.

3. Geo-spatial queries: Spatial data types (point, line, polygon, etc.) are treated as first-class citizens in ASTERIX, providing built-in support for geo-spatial (i.e., location-based) data. We will demonstrate, via a very simple Web-based user interface example (Figure 5), how ASTERIX might be used for spatial analysis of social data (from Twitter) to derive knowledge useful to society.

4. Fuzzy queries: An important use case for ASTERIX is querying, and analysis of semi-structured data drawn from Web sources (e.g., Twitter, social networking sites, event calendar sites, and so on), so it is a given that some of the incoming data will be dirty. Fuzzy matching is thus a key feature of ASTERIX that we plan to demonstrate. An example fuzzy matching query is shown in Figure 4(e). This example query is executed in parallel based on principles that we developed while studying how to perform fuzzy joins in the context of Hadoop [10].

5. Data Feeds: ASTERIX supports continuous data ingestion via data feeds and can be used to amass data from services such as Twitter or RSS feed-based services such as CNN news. We will demonstrate how ASTERIX can serve as an effective platform for archiving, tracking and analyzing social media activity.

6. STATUS OF ASTERIX

Currently, the ADM/AQL layer of ASTERIX is able to run parallel queries including lookups, large scans, parallel joins (regular and fuzzy), and parallel aggregates for data stored in partitioned LSM B+ trees and indexed via secondary indexes such as LSM-based R-trees. The system’s external data access and data feed features are also operational. We plan to offer a first open-source release of ASTERIX during the latter part of 2012, and we are now seeking early partners who would like to try ASTERIX on their favorite “Big Data” problems. Our ongoing work includes hardening and documenting the ASTERIX code base for initial public release, adding indexing support for fuzzy selection queries, improving the performance of spatial aggregation, adding support for continuous queries, extending AQL with windowing features, and starting to work with a few early users and use cases to learn by experience where we should go next.

Acknowledgements: This project is supported by NSF IIS awards 0910898, 0910859, 0910820, and 0844574, a grant from the UC Discovery program, and a matching donation from eBay.

7. REFERENCES