Big Data Platforms: What's Next?

Three computer scientists from UC Irvine address the question "What's next for big data?" by summarizing the current state of the big data platform space and then describing ASTERIX, their next-generation big data management system.

By Vinayak R. Borkar, Michael J. Carey, and Chen Li

DOI: 10.1145/2331042.2331057

A wealth of digital information is being generated daily, information has great potential value for many purposes if captured and aggregated effectively. In the past, data warehouses were largely an enterprise phenomenon, with large companies being unique in recording their day-to-day operations in databases, and warehousing and analyzing historical data to improve their businesses. Today, organizations and researchers in a wide variety of areas are seeing tremendous potential value and insight to be gained by warehousing the emerging wealth of digital information, popularly referred to as "big data," and making it available for analysis, querying, and other purposes [1].

Online businesses of all shapes and sizes are tracking customers' purchases, product searches, website interactions, and other information to increase the effectiveness of their marketing and customer service efforts. Governments and businesses are tracking the content of blogs and tweets to perform sentiment analysis. Public health organizations are monitoring news articles, tweets, and Web search trends to track the progress of epidemics. Social scientists are studying tweets and social networks to understand how information of various kinds spreads and/or how it can be more effectively utilized for the public good. It is no surprise that support for data-intensive computing, search, and information storage—a.k.a. big data analytics and management—is being touted as a critical challenge in today's computing landscape.

In this article we take a look at the current big data landscape, including its database origins, its more recent distributed systems rebirth, and current industrial practices and related trends in the research world. We then ask the question "What's next?" and provide a very brief tour of what one particular project, namely what our ASTERIX project, is doing in terms of exploring potential answers to this question (and why).

A BIT OF BIG DATA HISTORY

It is fair to say that the IT world has been facing big data challenges for over four decades—it's just that the definition of "big" has been changing. In the 1970s, big meant megabytes; over time, big grew to gigabytes and then to terabytes. Today, the IT notion of big has reached the petabyte range for conventional, high-end data warehouses, and exabytes are presumably waiting in the wings.

In the world of relational database systems, the need to scale databases to data volumes beyond the storage and/or processing capabilities of a single large computer system gave birth to a class of parallel database management systems known as "shared-nothing" parallel database systems [2]. As the name suggests, these systems run on networked clusters of computers, each with their own processors, memories, and disks. Data is spread over the cluster based on a partitioning strategy—usually hash partitioning, but sometimes range partitioning or random partitioning—and queries are processed by employing parallel, hash-based divide-and-conquer techniques.

A first generation of systems appeared in the 1980s, with pioneering prototypes from the University of Wisconsin and the University of Tokyo and the first commercial offering coming

Generative artwork by Gwen Vanhee using Craig Reynolds' algorithms for flocking behaviour.

from Teradata Corporation. The past decade has seen the emergence of a second major wave of these systems, with a number of startups delivering new parallel database systems that were then swallowed up through acquisitions by the industry's major hardware and software vendors. Because high-level, declarative language (SQL) front relational databases, users of parallel database systems have been shielded from the complexities of parallel programming. As a result, until quite recently, these systems have arguably been the most successful utilization of parallel computing.

During the latter 1990s, while the database world was admiring its "finished" work on parallel databases and major database software vendors were busy commercializing the results, the distributed systems world began facing its own set of big data challenges. The rapid growth of the World Wide Web and the resulting need to index and query its mushrooming content created big data challenges for search companies such as Inktomi, Yahoo, and Google. The processing needs in the search world were quite different, however, and SQL was not the answer, though shared-nothing clusters once again emerged as the hardware platform of choice. Google responded to these challenges by developing the Google File System (GFS), offering a familiar byte-stream-based file view of data that is randomly partitioned over hundreds or even thousands of nodes in a cluster [3]. GFS was then coupled with a programming model, MapReduce, to enable programmers to process big data by writing two user-defined functions, map and reduce [4]. The MapReduce framework applied these functions in parallel to individual data instances (Map) in GFS files and to sorted groups of instances that share a common key (Reduce)—similar to the partitioned parallelism used in shared-nothing parallel database systems. Yahoo and other big Web companies such as Facebook created an Apache open-source version of Google's big data stack, yielding the now highly popular Hadoop platform with its associated HDFS storage layer.

Similar to the big data back-end storage and analysis dichotomy, the historical record for big data also has a front-end (i.e., user-facing) story worth noting. As enterprises in the 1980s and 1990s began automating more and more of their day-to-day operations using databases, the database world had to scale up its online transaction processing (OLTP) systems as well as its data warehouses. Companies such as Tandem Computers responded with fault-tolerant, cluster-based SQL systems. Similarly,

but later in the distributed systems world, large Web companies were driven by very expansive user bases (up to tens or even hundreds of millions of Web users) to find solutions to achieve very fast simple lookups and updates to large, keyed data sets such as collections of user profiles. Monolithic SQL databases built for OLTP were rejected as being too expensive, too complex, and/or not fast enough, and today's "NoSQL movement" was born [5]. Again, companies such as Google and Amazon developed their own answers (BigTable and Dynamo, respectively) to meet this set of needs, and again, the Apache open-source community created corresponding clones (HBase and Cassandra, two of today's most popular and scalable key-value stores).

TODAY'S BIG DATA PLATFORM(S)

Hadoop and HDFS have grown to become the dominant platform for Big Data analytics at large Web companies as well as less traditional corners of traditional enterprises (e.g., for clickstream and log analyses). At the same time, data analysts have grown tired of the low-level MapReduce programming model, now choosing instead from among a handful of high-level declarative languages and frameworks that allow data analyses to be expressed much more easily and written and debugged much more quickly. These languages include Hive from Facebook (a variant of SQL) and Pig from Yahoo (a functional variant of the relational algebra, roughly). Tasks expressed in these languages are compiled down into a series of MapReduce jobs for execution on Hadoop clusters. Looking at workloads on real clusters, it has been reported that well over 60 percent of Yahoo's Hadoop jobs and more than 90 percent of Facebook's jobs now come from these higher-level languages rather than hand-written MapReduce jobs. MapReduce is essentially being relegated to the role of a big data runtime for higher-level, declarative data languages (which are not so very different than SQL).

Given this fact, it is interesting to analyze the pros and cons of MapReduce in this role as compared to more traditional parallel SQL runtime systems [6]. Important pros of Hadoop compared with parallel SQL systems include:

1. Open source availability versus expensive software licenses.
2. Multiple non-monolithic layers and components versus having only a top-level query API through which to access the data.
3. Support for access to file-based external data versus having to first design, load, and then index tables before being able to proceed.
4. Support for automatic and incremental forward recovery of jobs with failed tasks versus rolling long jobs back to their beginning to start all over again on failure.
5. Automatic data placement and rebalancing as data grows and machines come and go versus manual, DBA-driven data placement.
6. Support for replication and machine fail-over without operator intervention versus pager-carrying DBAs having to guide data recovery activities.

Some of the cons are:

1. Similar to early observations on why database systems' needs were not met by traditional OSs and their file systems [7], layering a record-based abstraction on top of a very large byte-sequential file abstraction leads to an impedance mismatch.
2. There is no imaginable reason, other than "because it is already there," to layer a high-level data language on top of a two-unary-operator runtime like MapReduce, as it can be quite unnatural (e.g., for joins) and can lead to suboptimal performance.
3. With random data block partitioning, the only available parallel query processing strategy is to "spray-and-pray" every query to all blocks of the relevant data files.
4. A flexible, semi-structured [8], schema-less data model (based on keys and values) means that important information about the data being operated on is known only to the programs operating on it (so program maintenance troubles await).
5. Coupling front- and back-end big data platforms to cover the full big data lifecycle requires significant use of bubble gum, baling wire, and hand-written ETL-like scripts.
6. While Hadoop definitely scales, its computational model is quite heavy (e.g., always sorting the data flowing between Map and Reduce,

Figure 1. ASTERIX Architecture

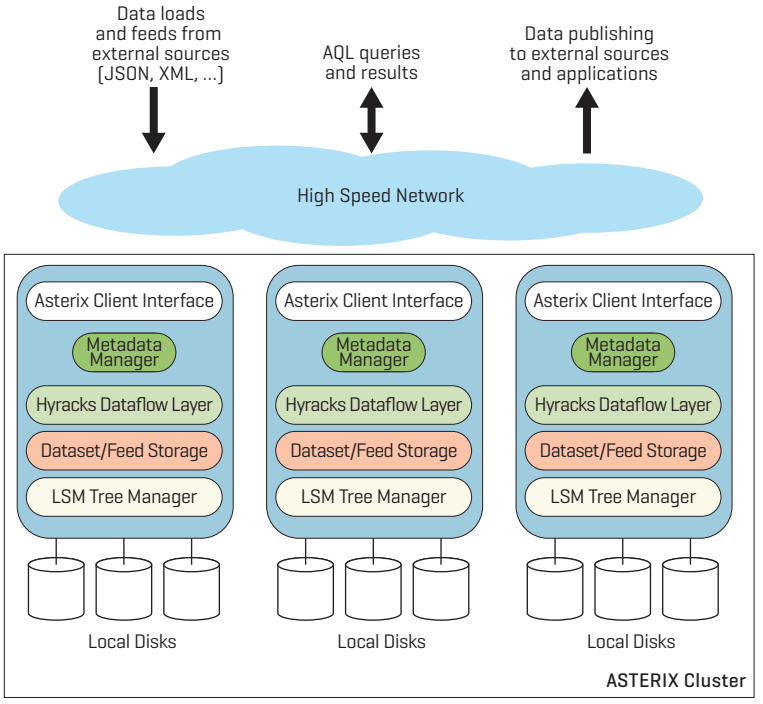


Figure 2. Example AQL schemas, queries, and results.

```
create type TweetMessageType as open {
  tweet_id: string,
  user:{
     screen-name: string,
     lang: string,
     friends_count: int32,
     statuses_count: int32,
     name: string,
     folllowers_count: int32
  },
  sender-location:_point ?
  send-time: datetime,
  referred_topics: {{string}}
  message-text: string
};

create dataset TweetMessges(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.

```
{{ {
    "tweetid": "1023",
    "user": {
        "screen-name": "dflynn24",
        "lang": "en",
        "friends_count": 46,
        "statuses_count": 987,
        "name": "danielle flynn",
        "followers_count": 47
    },
    "sender-location": "40.904177,-72.958996",
    "send-time": "2010-02-21T11:56:02-05:00",
    "referred-topics": {{ "verizon" }},
    "message-text": "i need a #verizon phone :("
  },
  {
    "tweetid": "1024",
    "user": {
        "screen-name": "miriamorous",
        "lang": "en",
        "friends_count": 69,
        "statuses_count": 1068,
        "name": "Miriam Songco",
        "followers_count": 78
    },
    "send-time": "2010-02-21T11:11:43-08:00",
    "referred-topics": {{ "commercials", "verizon", "att" }},
    "message-text": "#verizon & #att #commercials,
        so competitive"
  },
  {
    "tweetid": "1025",
    "user": {
        "screen-name": "dj33",
        "lang": "en",
        "friends_count": 96,
        "statuses_count": 1696,
        "name": "Don Jango",
        "followers_count": 22
    },
    "send-time": "2010-02-21T12:38:44-05:00",
    "referred-topics": {{ "verizon", "at&t", "iphone" }},
    "message-text": "I think I may switch from
        #verizon to #at&t"
  } }}
```

(d) Example tweet data

```
for $tweet1 in dataset('TweetMessages')
for $tweet2 in dataset('TweetMessages')
where $tweet1.tweetid != $tweet2.tweetid
    and $tweet1.message-text~=$tweet2.message-text
return {
    "tweet1-text": $tweet1.message-text,
    "tweet2-text": $tweet2.message-text
}
```

(e) An example fuzzy query over Tweet messages to find similar tweets

always persisting temporary data to HDFS between jobs in a multi-job query plan, etc. [9]).

WHAT'S NEXT?

Given the largely accidental nature of the current open-source Hadoop stack, and a need to store and manage as well as simply analyze data, we set out three years ago to design and implement a highly scalable platform for next-generation information storage, search, and analytics. We call the result a Big Data Management System (or BDMS). By combining and extending ideas drawn from semi-structured data management, parallel databases, and first-generation data-intensive computing platforms (notably Hadoop/HDFS), ASTERIX aims to be able to access, ingest, store, index, query, analyze, and publish very large quantities of semi-structured data. The design of the ASTERIX BDMS is well-suited to handling use cases that range all the way from rigid, relation-like data collections—whose structures are well understood and largely invariant—to flexible and more complex data, where little is planned ahead of time and the data instances are highly variant and self-describing.

Figure 1 provides an overview of a shared-nothing ASTERIX cluster and how its various software components map to cluster nodes. The bot-

ASTERIX aims to be able to access, ingest, store, index, query, analyze, and publish very large quantities of semi-structured data.

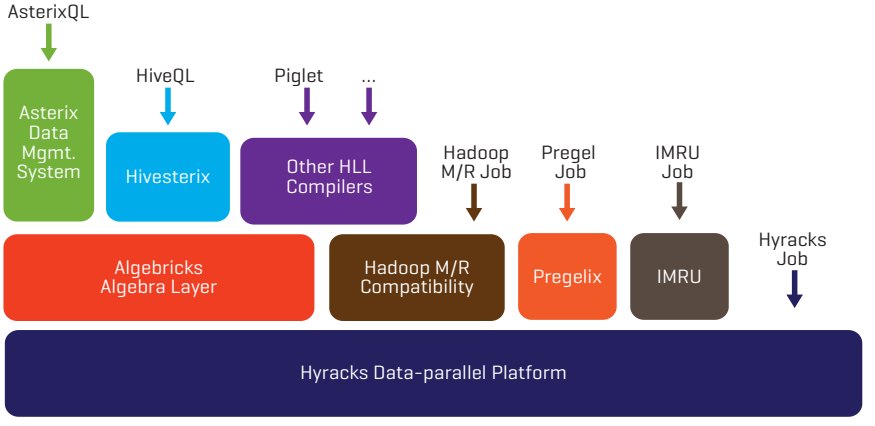
tom-most layer of ASTERIX provides storage capabilities for ASTERIX-managed datasets based on LSM-tree indexing (chosen in order to support high data-ingestion rates). Further up the stack is our data-parallel runtime, Hyracks [9], which sits at roughly the same level as Hadoop in implementations of Hive and Pig but supports a much more flexible computational model. The topmost layer of ASTERIX is a full parallel BDMS, complete with its own flexible data model (ADM) and query language (AQL) for describing, querying, and analyzing data. AQL is comparable to languages such as Pig or Hive, however ASTERIX supports native storage and indexing of data as well as having the ability to operate on externally resident data (e.g., data in HDFS).

The ASTERIX data model (ADM) borrowed data concepts from JSON and added more primitive types as well as type constructors from semi-structured and object databases. Figure 2(a) illustrates ADM by showing how it might be used to define a record type for modeling Twitter messages. The record type shown is an open type, meaning that its instances should conform to its specification but will be allowed to contain arbitrary additional fields that can vary from one instance to another. The example also shows how ADM includes features such as optional fields with known types (e.g., "sender-location"), nested collections of primitive values ("referred-topics"), and nested records ("user"). More information about ADM can be found in a recent paper that provides an overview of the ASTERIX project [10].

Figure 2(d) shows an example of how a set of TweetMessageType instances would look. Data storage in ASTERIX is based on the concept of a dataset, a declared collection of instances of a given type. ASTERIX supports both system-managed datasets—such as the TweetMessages dataset declared at the bottom of Figure 2(a)—which are stored and managed by ASTERIX as partitioned, LSM-based B+ trees with optional secondary indexes, and external datasets, whose data can reside in existing HDFS files or collections of files in the cluster nodes' local file systems.

The ASTERIX query language is called AQL, a declarative query language designed by borrowing the essence of the XQuery language, most notably its FLWOR expression constructs and composability, and then simplifying and adapting it to the

Figure 3. ASTERIX software stack.



types and data modeling constructs of ADM. Figure 2(b) illustrates AQL by example. This AQL query runs over the TweetMessages dataset to compute, for those tweets mentioning "verizon," the number of tweets that refer to each topic appearing in those tweets. Figure 2(c) shows the results of this example query when run against the sample data of Figure 2(d).

One of the primary application areas envisioned for ASTERIX is warehouse-based Web data integration [11]. As such, ASTERIX comes "out of the box" with a set of interesting capabilities that we feel are critical for such use cases. One is built-in support for a notion of data feeds to continuously ingest, pre-process, and persist data from external data sources such as Twitter. Another is support for fuzzy selection and fuzzy (a.k.a. set-similarity) joins, as Web data and searches are frequently ridden with typos and/or involve sets (e.g., of interests) that should be similar but not identical. Figure 2(e) illustrates a fuzzy join query in AQL. Yet another built-in capability is basic support for spatial data (e.g., locations of mobile users) and for queries whose predicates include spatial criteria.

Figure 3 shows the nature of the open-source ASTERIX software stack, which supports the ASTERIX system but also aims to address other big data requirements. To process queries such as the example from Figure 2(b), ASTERIX compiles each AQL query into an Algebricks algebraic program. This program is then optimized via algebraic rewrite rules that reorder the Algebricks operators as well as introduce partitioned parallelism for scalable execution, after which code generation translates the resulting physical query plan into a corresponding Hyracks job that uses Hyracks to compute the desired query result. The left-hand side of Figure 3 shows this layering. As also indicated in the figure, the Algebricks algebra layer is data-model-neutral and is therefore also able to support other high-level data languages (such as a Hive port that we have built).

The ASTERIX open-source stack also offers a compatibility layer for users with Hadoop jobs who wish to run them using our software as well as offering several other experimental big data programming packages (including Pregelix, a Pregel-like layer that runs on Hyracks, and IMRU, an iterative map/reduce/update layer that targets large-scale machine learning applications [12]).

ASTERIX GOING FORWARD

Currently the ADM/AQL layer of ASTERIX can run parallel queries including lookups, large scans, parallel joins (both regular and fuzzy), and parallel aggregates. Data is stored natively in partitioned B+ trees and can be indexed via secondary indexes such as B+ trees, R-trees, or inverted indexes. The system's external data access and data feed features are also operational. We plan to offer a first open-source release of ASTERIX in late 2012, and we are seeking a few early partners who would like to run ASTERIX on their favorite big data problems. Our ongoing work includes preparing the code base for an initial public release, completing our initial transaction story, adding additional indexing support for fuzzy queries, and providing a key-value API for applications that prefer a "NoSQL" style API over a more general query-based API. More information about the project and its current code base can be found on our project website (http://asterix.ics.uci.edu/).

It is worth pointing out ASTERIX is a counter-cultural project in several ways. First, rather than "tweaking" Hadoop or other existing packages, we set out to explore the big data platform space from the ground up. We are learning a great deal from doing so, as it is surprising just how many interesting engineering and research problems are still lurking in places related to "things that have already been done before." Second, rather than building a highly-specialized system to later be glued into a patchwork of such systems, we are exploring the feasibility of a "one size fits a bunch" system that addresses a broader set of needs (e.g., by offering data storage and indexing as well as support for external data analysis, short- and medium-sized query support as well as large batch jobs, and a key-value API as well as a query-based one).

Acknowledgments

The ASTERIX project has been supported so far by NSF IIS awards 0910989, 0910859, 0910820, and 0844574, a grant from the UC Discovery program, a matching donation from eBay, a Facebook Fellowship, and a donation of cluster access time from Yahoo! Research.

References

[1] Lohr, S. The age of big data. *The New York Times*. February 12, 2012.

[2] DeWitt, D. and Gray, J. Parallel database systems: The future of high performance database systems. *Communications of the ACM* 35, 6 (June 1992).

[3] Ghemawat, S., Gobioff, H., and Leung, S. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems* (Lake George, Oct. 2003), 29-43.

[4] Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Sixth USENIX Symposium on Operating Systems Design and Implementation* (San Francisco, December 2004), 137-150.

[5] Cattell, R. Scalable SQL and NoSQL data stores. *SIGMOD Record* 39, 4 (May 2011).

[6] Borkar, V., Carey, M., and Li, C. Inside "big data management": Ogres, onions, or parfaits. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)* (Berlin, March 2012), 3-14.

[7] Stonebraker, M. Operating system support for database management. *Communications of the ACM* 24, 7 (July 1981).

[8] Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 1999.

[9] Borkar, V., Carey, M., Grover, R., Onose, N., and Vernica, R. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Proceedings of the 2011 IEEE International Conference on Data Engineering (ICDE)* (Hannover, Germany, April 2011), 1151-1162.

[10] A. Behm, A., Borkar, V., Carey, M., Grover, R., Li, C., Onose, N., Vernica, R., Deutsch, A., Papakonstantinou, Y., and Tsotras, V. ASTERIX: Towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases* 29, 3 (2011).

[11] Alsubaiee, S., Behm, A., Grover, R., Vernica, R., Borkar, V., Carey, M., and Li, C. ASTERIX: Scalable Warehouse-Style Web Data Integration. In *Proceedings of the Ninth International Workshop on Information Integration on the Web (IIWeb)* (Phoenix, May 2012).

[12] Malewicz, G., Austern, M., Bik, A., Dehnert, J., Horn, I., Leiser, N., and G. Czajkowski, G. Pregel: A system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Indianapolis, June 2010), 135-146.

Biographies

Vinayak R. Borkar is an assistant project scientist and part-time computer science Ph.D. student in the Bren School of Information and Computer Sciences at UC Irvine. He has an M.S. in computer science and engineering from IIT Bombay.

Michael J. Carey is a professor in the Bren School of Information and Computer Sciences at UC Irvine. He has a Ph.D. in computer science from UC Berkeley.

Chen Li is an associate professor in the Bren School of Information and Computer Sciences at UC Irvine. He has a Ph.D. in computer science from Stanford University.