Introducing
*Asterix*<sup>DB</sup>

**(A Next-Generation Big Data Management System)**

Michael Carey (CMU EE ’79, ’81)

*Information Systems Group*

*CS Department*

*UC Irvine*

http://isg.ics.uci.edu
Rough Plan

• Context (a brief history of 2 worlds)
• AsterixDB: a next-generation BDMS
• The ASTERIX open software stack
  – AsterixDB: Big Data Management 2.0
  – Hivesterix: HiveQL on Algebricks
  – Pregelix: Pregel on Hyracks
  – IMRU: Big ML on Hyracks
• “One Size Fits a Bunch” (and Q&A)
Everyone’s Talking About Big Data

- Driven by unprecedented growth in data being generated and its potential uses and value
  - Tweets, social networks (statuses, check-ins, shared content), blogs, click streams, various logs, ...
  - Facebook: > 845M active users, > 8B messages/day
  - Twitter: > 140M active users, > 340M tweets/day
Big Data / Web Warehousing
Big Data in the *Database* World

• Enterprises wanted to store and query historical business data (data warehouses)
  – 1970’s: Relational databases appeared (w/SQL)
  – 1980’s: Parallel database systems based on “shared-nothing” architectures (Gamma, GRACE, *Teradata*)
  – 2000’s: Netezza, Aster Data, DATAllegro, Greenplum, Vertica, ParAccel, ... (Serious “Big $” acquisitions!)

Each node runs an instance of an indexed, DBMS-style data storage and runtime system
Also OLTP Databases

- On-line transaction processing is another key Big Data dimension
  - OLTP applications power daily business
  - Producers of the data being warehoused
- Shared-nothing also a serious architecture for OLTP
  - 1980’s: Tandem’s NonStop SQL
Parallel Database Software Stack

Notes:
• One storage manager per machine in a parallel cluster
• Upper layers orchestrate their shared-nothing cooperation
• One way in/out: through the SQL door at the top
Big Data in the *Systems* World

- Late 1990’s brought a need to index and query the rapidly exploding content of the Web
  - DB technology tried but failed (*e.g.*, Inktomi)
  - Google, Yahoo! *et al* needed to do something
- Google responded by laying a new foundation
  - Google File System (GFS)
    - OS-level byte stream files spanning 1000’s of machines
    - Three-way replication for fault-tolerance (availability)
  - MapReduce (MR) programming model
    - User functions: Map and Reduce (and optionally Combine)
    - “*Parallel programming for dummies*” – MR runtime does the heavy lifting via partitioned parallelism
Soon a Star Was Born...

- Yahoo!, Facebook, and friends cloned Google’s “Big Data” infrastructure from papers
  - GFS $\rightarrow$ Hadoop Distributed File System (HDFS)
  - MapReduce $\rightarrow$ Hadoop MapReduce
  - In wide use for Web indexing, click stream analysis, log analysis, information extraction, some machine learning

- Tired of problem-solving with just two unary operators, higher-level languages were developed to hide MR
  - Pig (Yahoo!), Jaql (IBM), Hive (Facebook)
  - Now in heavy use over MR (Pig > 60%, HiveQL > 90%)

- Similar happenings at Microsoft
  - Cosmos, Dryad, DryadLINQ, SCOPE (powering Bing)
Also Key-Value Stores

• Another Big Data dimension, for applications powering social sites, gaming sites, and so on
  – Systems world’s version of OLTP, roughly

• Need for simple record stores
  – Simple, key-based retrievals and updates
  – Fast, highly scalable, highly available

• Numerous “NoSQL” systems (see Cattell survey)
  – Proprietary: BigTable (Google), Dynamo (Amazon), …
  – Open Source: HBase (BigTable), Cassandra (Dynamo), …
Open Source Big Data Stack

Notes:
- Giant byte sequence files at the bottom
- Map, sort, shuffle, reduce layer in middle
- Possible storage layer in middle as well
- Now at the top: HLL’s

(Huh...?)
Existing Solution(s)
AsterixDB: “One Size Fits a Bunch”

Semistructured Data Management

Parallel Database Systems

Data-Intensive Computing
• Build a new Big Data Management System (BDMS)
  – Run on large commodity clusters
  – Handle mass quantities of semistructured data
  – Openly *layered*, for selective reuse by others
  – Share with the community via open source (June 2013)

• Conduct scalable information systems research
  – Large-scale query processing and workload management
  – Highly scalable storage and index management
  – Fuzzy matching, spatial data, date/time data (all in parallel)
  – Novel support for “fast data” (both in and out)

Train next generation of “Big Data” graduates
ASTERIX Hadoop Influences

• Open source availability ("price is right")
• Non-monolithic layers or components
• Support for external data access (in files)
• Roll-forward recovery of jobs on failures
• Automatic data placement, migration, replication
AsterixDB System Overview

Data loads and feeds from external sources

Hi-Speed Interconnect

AQL queries/results

Data publishing

Asterix Cluster

Asterix Client Interface
AQL Compiler
Metadata Manager
Hyracks Dataflow Engine
Dataset / Feed Storage
LSM Tree Manager

(ADM = ASTERIX Data Model; AQL = ASTERIX Query Language)
create dataverse LittleTwitterDemo;

create type TweetMessageType as open {
  tweetid: 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 tweetid;

Highlights:
• JSON++ based data model
• Rich type support (spatial, temporal, ...)
• Records, lists, bags
• Open vs. closed types
• External data sets and datafeeds
Ex: TweetMessages Dataset

{{
  "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 like nowwww! :(("
},
{
  "tweetid": "1024",
  "user": {
    "screen-name": "miriamorous",
    "lang": "en",
    "friends_count": 69,
    "statuses_count": 1068,
    "name": "Miriam Songco",
    "followers_count": 78
  },
  "send-time": "2010-02-21T11:43:08-00",
  "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": [{"charlore"}],
  "message-text": "Chillin at dca waiting for 900am flight to #charlore and from there to providenciales"
},
{
  "tweetid": "1026",
  "user": {
    "screen-name": "reallyleila",
    "lang": "en",
    "friends_count": 106,
    "statuses_count": 107,
    "name": "Leila Samii",
    "followers_count": 52
  },
  "send-time": "2010-02-21T21:31:57-06:00",
  "referred-topics": [{"verizon", "at&t", "iphone"}],
  "message-text": "I think a switch from #verizon to #at&t may be in my near future... my smartphone is like a land line compared to the #iphone!"
}}

Ex: TweetMessages Dataset
Ex: List the topics being Tweeted about, along with their associated Tweet counts, in Verizon-related Tweets:

```plaintext
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)
}
```
Fuzzy Joins in AQL

- **Ex: Find Tweets with similar content:**

  ```
  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
  }
  ```

- **Or:** Find Tweets about similar topics:

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

- **Ex:** Create “Fast Data” feeds for Tweets and News articles:

  ```
  create feed dataset TweetMessages(TweetMessageType)
  using TwitterAdapter ("interval"="10")
  apply function addHashTagsToTweet
  partitioned by key tweetid;

  create feed dataset NewsStories(NewsType)
  using CNNFeedAdapter ("topic"="politics","interval"="600")
  apply function getTaggedNews
  partitioned by key storyid;

  create index locationIndex on Tweets(sender-location) type rtree;

  begin feed TweetMessages;  
  begin feed NewsStories;
  ```

**Highlights:**
- Philosophy: “keep everything”
- Data ingestion, not data streams
- Previous queries unchanged
The ASTERIX Software Stack

AsterixQL

HiveQL

Piglet

... Other HLL Compilers

Hadoop M/R Job

Pregel Job

IMRU Job

Algebricks Algebra Layer

Hadoop M/R Compatibility

Pregelix

IMRU

Hyracks Job

Hyracks Data-parallel Platform
Algebricks

- Set of (data model agnostic) logical operations
- Set of physical operations
- Rewrite rule framework (logical, physical)
- Generally applicable rewrite rules (including parallelism)
- Metadata provider API (catalog info for Algebricks)
- Mapping of physical operations to Hyracks operators
• Partitioned-parallel platform for data-intensive computing
• Job = dataflow DAG of operators and connectors
  – Operators consume and produce *partitions* of data
  – Connectors *route* (repartition) data between operators

• **Hyracks vs. the “competition”**
  – Based on time-tested parallel database principles
  – vs. Hadoop: More flexible model and less “pessimistic”
  – vs. Dryad: Supports data as a first-class citizen
Hyracks (cont.)

{NC1: cust1.dat}
{NC2: cust2.dat}

Scanner (CUSTOMER)
E1[hash(C_CUSTKEY)]

HashJoin
C_CUSTKEY = O_CUSTKEY
E2[hash(O_CUSTKEY)]

{NC3: ord1.dat, NC2: ord1.dat}
{NC1: cust2.dat, NC5: ord2.dat}

HashGroupby
C_MKTSEGMENT
Agg: count(O_ORDKEY)
E3[hash(C_MKTSEGMENT)]

Writer

E4[1:1]
Hyracks Performance
(On a small cluster with 40 cores & 40 disks in 2011)

(Faster 😊)
Hadoop M/R Compatibility

File write

UDAF(Reduce)

UDAF(Combine)

UDF (Map)

File scan

HDFS

Reduce Tasks

Combiners

Map Tasks
Hyracks Performance Benefits*

- **K-Means (Hadoop M/R compatibility layer)**
  - Push-based (eager) job activation
  - Default sorting/hashing is on serialized (binary) data
  - Pipelining (w/o disk I/O) between Mapper and Reducer
  - Relaxed connector semantics exploited at network level

- **TPC-H Query (in addition to the above)**
  - Hash-based join strategy doesn’t require sorting or artificial data multiplexing and de-multiplexing
  - Hash-based aggregation is similarly more efficient

- **Fault-Tolerant TPC-H Experiment**
  - Faster → smaller failure target, more affordable retries
  - Need incremental recovery, but not blind pessimism

*Proc. 27th ICDE Conf.*, Hannover, Germany, April 2011.
Hivesterix (HiveQL on Hyracks)

• Replace Hive’s runtime: Hadoop → Hyracks
  – One DAG Hyracks job per HiveQL query
  – More algorithm choices for joining, grouping, sorting
  – Binary data representation

• Build Hive on Algebricks
  – More optimization opportunities, e.g., data properties
  – *Goal*: Validate the Algebricks value proposition

• Reuse Hive package as a library
  – Semantic analysis, optimizations, UDFs, types, formats, and SerDes are all reused (to potentially track Hive)
  – Input data comes from HDFS in Hive’s native format
Hivesterix (cont.)

Query

Hive

Hive compiler

Hive UDF

Hive format, SerDe

Hive UDF FuncJon evaluator

Hash funcJons, comparators...

Datasource

Hive format, SerDe

Hive UDF

Hive 3.9X faster (avg)

TPC-H 500X, 97 machine cluster (4 partitions/machine)

Hivesterix(4 partitions/node)  Hive(4m4r/node)

Time (seconds)

Query

Hivesterix 3.9X faster (avg)
Pregelix (Joint w/Yahoo!)

Think like a Vertex:
- Pregel
- Giraph
- GraphLab

... 

- Task scheduling
- Memory management
- Message delivery
- Network management

Pregelx Semantics

Pregelix

UDF(compute)

Barrier

Msg

Vertex

Pregel Semantics

Hyracks parallel dataflow engine

GraphLab

Message delivery

Task scheduling

Memory management

Network management

Think like a Vertex:
- Pregel
- Giraph
- GraphLab

......

- Task scheduling
- Memory management
- Message delivery

Hyracks parallel dataflow engine
IMRU (Joint w/UCSC, Yahoo!)

- Iterative Map-Reduce-Update programming model
- Useful for iterative hill-climbing ML (e.g., Batch Gradient Descent)

Experiment: 2.3 B vectors from advertising domain, about 37.1B features. Used 30 machines to compare Hyracks with Vowpal Wabbit (VW): 114.54s on Hyracks vs. 124.41s on VW
Why AsterixDB?

• “One Size Fits a Bunch” can offer better functionality, manageability, and performance than gluing together multiple point solutions (e.g., Hadoop + Hive + MongoDB):
  – LSM indexes for dynamic data with queries
  – Spatial indexing and spatial query capabilities
  – Fuzzy indexing and query processing for similarity
  – External datasets (and datafeeds) for external data
  – Powerful graph-processing module: Pregelix

• Hyracks is a more powerful, flexible, and efficient run-time dataflow engine than Hadoop – and supports an open stack
  – Operators/primitives based on parallel DBMS best practices
  – Experiments show up to 10x performance speedups at scale (on disk-resident problems and data sizes)
Current Status

• Approaching 4 years of initial NSF project (~250 KLOC)
• Code scale-tested on a 6-rack Yahoo! Labs cluster with roughly 1400 cores and 700 disks
• Hyracks and Pregelix ready now, IMRU very soon
• AsterixDB BDMS: Beta out in June! (June 6th, 2013)
  – Semistructured “NoSQL” style data model
  – Declarative parallel queries, inserts, deletes, ...
  – LSM-based storage/indexes (primary & secondary)
  – Internal and external datasets both supported
  – Fuzzy and spatial query processing
  – NoSQL-like transactions (for inserts/deletes)
Partial Cast List

- Faculty and research scientists
  - UCI: Michael Carey, Chen Li; Vinayak Borkar, Nicola Onose (Google)
  - UCR: Vassilis Tsotras
- PhD students
  - UCI: Rares Vernica (HP Labs), Alex Behm (Cloudera), Raman Grover, Yingyi Bu, Sattam Alsubaiee, Yassar Altowim, Hotham Altwajry, Pouria Pirzadeh, Zachary Heilbron, Young-Seok Kim
  - UCR/UCSD: Jarod Wen, Preston Carman, Nathan Bales (Google)
- MS students
  - UCI: Guangqiang Li (MarkLogic), Vandana Ayyalasomayajula (Yahoo!), Siripen Pongpaichet, Ching-Wei Huang, Manish Honnatti (Zappos), Xiaoyu Ma, Madhusudan Cheelangi (Google), Khurram Faraaz (IBM DB2), Tejas Patel
- BS students (alumni)
  - UCI: Roman Vorobyov, Dustin Lakin
- Foreign affiliates
  - Thomas Bodner (T.U. Berlin), Markus Dressler (HPI), Rico Bergmann (Humboldt U.)
Collaborators

• Facebook
  – Funded Facebook Fellowship
  – Provided access to Hive data warehouse workload information

• Yahoo! Research
  – Hyracks as infrastructure for ScalOps language
  – Provided access to 200-node cluster (and data)
  – Funded 3 Key Scientific Challenge graduate student awards

• Rice University
  – Hyracks for online aggregation

• UC Santa Cruz
  – Hyracks for IMRU programming model
  – Added support for asynchronous fixpoint computations

• UC San Diego
  – Added fine-grained lineage capture capability to Hyracks

• Apache Software Foundation
  – Hyracks/Algebricks as foundation for parallel XQuery engine
  – One student funded by Google Summer of Code 2012.
For More Info

NSF project page: http://asterix.ics.uci.edu

Open source code base:

- ASTERIX: http://code.google.com/p/asterixdb/
- Pregelix: http://hyracks.org/projects/pregelix/
Questions...?