Big NoSQL Data

Mike Carey
mjcarey@ics.uci.edu
Announcements

• Keep watching the course wiki page (especially its attachments):
  • https://grape.ics.uci.edu/wiki/asterix/wiki/stats170ab-2018

• Ditto for the Piazza page (for Q&A):
  • http://piazza.com/uci/winter2018/stats170a/home

• HW #4 is underway:
  • Expect getting 10K Tweets to take ~1 hour
  • Leave time for wrangling and rate-limiting “fun”
    • Id data type options: bigint or varchar (varchar may be easier)
    • Suggestion: get list of Tweets, then go straight to PostgreSQL
  • Any initial questions? (Don’t start this one late!)
Plan for Today’s Lecture

• The pre-relational and relational eras
• Moving beyond rows and columns (?)
  1. The object-oriented DB era
  2. The object-relational DB era
  3. The XML DB era
  4. The NoSQL DB era
• AsterixDB as a Big NoSQL Data exemplar
  • User model (DDL, DML, etc.)
  • System architecture and internal highlights
• Conclusions and class participation (😊)
The Birth of Today’s DBMS Field

• In the beginning was the Word, and the Word was with Codd, and the Word was Codd...
  • 1970 CACM paper: “A relational model of data for large shared data banks”

• Many refer to this as the first generation of (real?) database management systems
The First Decade B.C.

• The need for a data management library, or a database management system, had actually been well recognized
  • Hierarchical DB systems (e.g., IMS from IBM)
  • Network DB systems (most notably CODASYL)
• These systems provided navigational APIs
  • Systems provided files, records, pointers, indexes
  • Programmers had to (carefully!) scan or search for records, follow parent/child structures or pointers, and maintain code when anything physical changed
The First Decade B.C. (cont.)

Order (id, custName, custCity, total)

Product (sku, name, listPrice, size, power)

Item (ino, qty, price)

Item-Order

Item-Product

(Parent ← child record sets)
Enter the Relational DB Era

Order \((id,\ custName,\ custCity,\ total)\)

| 123 | Fred | LA | 25.97 |

Item \((order-id, ino,\ product-sku, qty, price)\)

| 123 | 1 | 401 | 2 | 9.99 |
| 123 | 2 | 544 | 1 | 3.99 |

Product \((sku,\ name, listPrice, size, power)\)

| 401 | Garfield T-Shirt | 9.99 | XL | null |
| 544 | USB Charger | 5.99 | null | 115V |

- Be sure to notice that
  - Everything’s now (logical) rows and columns
  - The world is flat; columns are atomic (1NF)
  - Data is now connected via keys (foreign/primary)
As the Relational Era Unfolded

• The Spartan simplicity of the relational data model made it possible to start tackling the opportunities and challenges of a *logical data model*
  • Declarative queries (Rel Alg/Calc, Quel, QBE, SQL, ...)
  • Transparent indexing (physical data independence)
  • Query optimization and execution
  • Views, constraints, referential integrity, security, ...
  • Scalable (shared-nothing) parallel processing

• Today’s multi-$B$ industry was slowly born
  • Commercial adoption took ~10-15 years
  • Parallel DB systems took ~5 more years
Enter the Object-Oriented DB Era

- Notice that:
  - Data model contains objects and pointers (OIDs)
  - The world is no longer flat – the Order and Product schemas now have set(Item) and Product in them, respectively
What OODBs Sought to Offer

• Motivated largely by late 1980’s CAx applications (e.g., mechanical CAD, VLSI CAD, software CAD, ...)
  • Rich schemas with inheritance, complex objects, object identity, references, ... 
  • Methods (“behavior”) as well as data in the DBMS
  • Tight bindings with (OO) programming languages
  • Fast navigation, some declarative querying

• Ex: Gemstone, Ontos, Objectivity, Versant, Object Design, O2, also DASDBS (sort of)
Why OODBs “Fell Flat”

• Too soon for another (radical) DB technology
  • Also technically immature relative to RDBMSs
• Tight PL bindings were a double-edged sword
  • Data shared, outlives programming languages
  • Bindings led to significant system heterogeneity
  • Also made schema evolution a major challenge
• Systems “overfitted” in some dimensions
  • Inheritance, version management, ...
  • Focused on thick clients (e.g., CAD workstations)
Enter the Object-Relational DB Era

<table>
<thead>
<tr>
<th>Order (id, customer, total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item (order-id, ino, product-sku, qty, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(123)</td>
</tr>
<tr>
<td>(123)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product (sku, name, listPrice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClothingProduct (size) under Product</td>
</tr>
<tr>
<td>ElectricProduct (power) under Product</td>
</tr>
<tr>
<td>401</td>
</tr>
<tr>
<td>544</td>
</tr>
</tbody>
</table>

- Be sure to notice:
  - “One size fits all!”
  - UDTs/UDFs, table hierarchies, references, ...
  - But the world got flatter again...
    (Timing lagged OODBs by just a few years)
What O-R DBs Sought to Offer

• Motivated by newly emerging application opportunities (multimedia, spatial, text, ...)
  • *User-defined functions (UDTs/UDFs) & aggregates*
  • Data blades (UDTs/UDFs + indexing support)
  • OO goodies for tables: row types, references, ...
  • Nested tables (well, at least Oracle added these)

• Back to a model where applications were loosely bound to the DBMS (e.g., ODBC/JDBC)

• *Ex: ADT-Ingres, Postgres, Starburst, UniSQL, Illustra, DB2, Oracle*
Why O-R DBs “Fell Flat”

• Significant differences across DB vendors
  • SQL standardization lagged somewhat
  • Didn’t include details of UDT/UDF extensions
  • Tough to extend the innards (for indexing)

• Application issues (and multiple platforms)
  • Least common denominator vs. coolest features
  • Tools (e.g., DB design tools, ORM layers, …)

• Also still probably a bit too much too soon
  • IT departments still rolling in RDBMSs and creating relational data warehouses
Then Came the XML DB Era

<Order id="123">
  <Customer>
    <custName>Fred</custName>
    <custCity>LA</custCity>
  </Customer>
  <total>25.97</total>
  <Items>
    <Item ino="1">
      <product-sku>401</product-sku>
      <qty>2</qty>
      <price>9.99</price>
    </Item>
    <Item ino="2">
      <product-sku>544</product-sku>
      <qty>1</qty>
      <price>3.99</price>
    </Item>
  </Items>
</Order>

<Product sku="401">
  <name>Garfield T-Shirt</name>
  <listPrice>9.99</listPrice>
  <size>XL</size>
</Product>

<Product sku="544">
  <name>USB Charger</name>
  <listPrice>5.99</listPrice>
  <power>115V</power>
</Product>

Note that
- The world’s less flat again
- We’re now in the 2000’s
What XML DBs Sought to Offer

• One <flexible/> data model fits all (XML)
  • Origins in document markup (SGML)
  • Nested data
  • Schema variety/optionality

• New declarative query language (XQuery)
  • Designed both for querying and transformation
  • Early standardization effort (W3C)

• Two different DB-related use cases, in reality
  • Data storage: Lore (pre-XML), Natix, Timber, Ipedo, MarkLogic, BaseX; also DB2, Oracle, SQL Server
  • Data integration: Nimble Technology, BEA Liquid Data (from Enosys), BEA AquaLogic Data Services Platform
Why XML DBs “Fell Flat” Too

• Document-centric origins (vs. data use cases) of XML Schema and XQuery made a mess of things
  • W3C XPATH legacy (😊)
  • Document identity, document order, ...
  • Attributes vs. elements, nulls, ...
  • Mixed content (overkill for non-document data)
• Two other external trends also played a role
  • SOA and Web services came but then went
  • JSON (and RESTful services) appeared on the scene
• Note: Likely still an important niche market...
Now the “NoSQL” DB Era

• *Not* from the DB world!
  • Distributed systems folks
  • Also various startups

• From caches → K/V use cases
  • Needed massive scale-out
  • OLTP (vs. parallel DB) apps
  • Simple, low-latency API
  • Need a key K, but want no schema for V
  • Record-level atomicity, replica consistency varies
NoSQL Data (JSON-based)

Collection(Order)

```json
{"id": "123",
 "Customer": {
  "custName": "Fred",
  "custCity": "LA"
 }
 "total": 25.97,
 "Items": [
  {
   "product-sku": 401,
   "qty": 2,
   "price": 9.99
  },
  {
   "product-sku": 544,
   "qty": 1,
   "price": 3.99
  }
 ]
}
```

Collection(Product)

```json
{"sku": 401,
 "name": "Garfield T-Shirt",
 "listPrice": 9.99,
 "size": "XL"
 }

{"sku": 544,
 "name": "USB Charger",
 "listPrice": 5.99,
 "power": "115V"
 }
```

Note that
- The world’s not flat, but it’s less <messy/>
- We’re now in the 2010’s, timing-wise
Current NoSQL Trends

- Popular examples: MongoDB, Couchbase
- Coveting the benefits of many DB goodies
  - Secondary indexing and non-key access
  - Declarative queries
  - Aggregates and now (initially small) joins
- Seem to be heading towards...
  - **BDMS** (think scalable, OLTP-aimed, parallel DBMS)
  - Declarative queries and query optimization, but applied to schema-less data
  - Return of (some, optional!) schema information
Our Answer: Apache AsterixDB

ASTERIX Goal:
To ingest, digest, persist, index, manage, query, analyze, and publish massive quantities of semistructured information...

http://asterixdb.apache.org/
Just How **Big** is “Big Data”?

This is big data!
AsterixDB: “One Size Fits a Bunch”

Semistructured Data Management

BDMS Desiderata:
- Able to manage data
- Flexible data model
- Full query capability
- Continuous data ingestion
- Efficient and robust parallel runtime
- Cost proportional to task at hand
- Support “Big Data data types”

Parallel Database Systems

1st Generation “Big Data” Systems
CREATE DATAVERSE TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
    id: int,
    alias: string,
    name: string,
    userSince: datetime,
    friendIds: {{ int }},
    employment: [EmploymentType]
};

CREATE TYPE EmploymentType AS {
    organizationName: string,
    startDate: date,
    endDate: date?
};

CREATE DATASET GleambookUsers (GleambookUserType)
PRIMARY KEY id;

Highlights include:
• JSON++ based data model
• Rich type support (spatial, temporal, ...)
• Records, lists, bags
• Open vs. closed types
ASTERIX Data Model (ADM)

CREATE DATaverse TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
   id: int
};

CREATE TYPE EmploymentType AS {
   organizationName: string,
   startDate: date,
   endDate: date?
};

CREATE DATASET GleambookUsers
   (GleambookUserType)
PRIMARy KEY id;

Highlights include:
• JSON++ based data model
• Rich type support (spatial, temporal, …)
• Records, lists, bags
• Open vs. closed types
ASTERIX Data Model (ADM)

CREATE DATAVVERSE TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
    id: int
}

CREATE TYPE GleambookMessageType AS {
    messageId: int,
    authorId: int,
    inResponseTo: int?,
    senderLocation: point?,
    message: string
}

CREATE TYPE EmploymentType AS {
    organizationName: string,
    startDate: date,
    endDate: date?
}

CREATE DATASET GleambookUsers (GleambookUserType)
    PRIMARY KEY id;

CREATE DATASET GleambookMessages (GleambookMessageType)
    PRIMARY KEY messageId;

Highlights include:
- JSON++ based data model
- Rich type support (spatial, temporal, ...)
- Records, lists, bags
- Open vs. closed types
Ex: GleambookUsers Data

{
"id":1, "alias":"Margarita", "name":"MargaritaStoddard", "nickname":"Mags",
"userSince":datetime("2012-08-20T10:10:00"), "friendIds":{{2,3,6,10}},
"employment": [{"organizationName":"Codetechno", "startDate":date("2006-08-06")},
    {
"organizationName":"geomedia",  "startDate":date("2010-06-17"),
    "endDate":date("2010-01-26")} },
"gender":"F"
},

{"id":2, "alias":"Isbel", "name":"IsbelDull", "nickname":"Izzy",
"userSince":datetime("2011-01-22T10:10:00"), "friendIds":{{1,4}},
"employment": [{"organizationName":"Hexviafind", "startDate":date("2010-04-27")} ]
},

{"id":3, "alias":"Emory", "name":"EmoryUnk",
"userSince":datetime("2012-07-10T10:10:00"), "friendIds":{{1,5,8,9}},
"employment": [{
"organizationName":"geomedia",  "startDate":date("2010-06-17"),
    "endDate":date("2010-01-26")} ]
},

......
Other DDL Features

CREATE INDEX gbUserSinceIdx ON GleambookUsers(userSince);
CREATE INDEX gbAuthorIdx ON GleambookMessages(authorId) TYPE BTREE;
CREATE INDEX gbSenderLocIndex ON GleambookMessages(senderLocation) TYPE RTREE;
CREATE INDEX gbMessageIdx ON GleambookMessages(message) TYPE KEYWORD;

CREATE TYPE AccessLogType AS CLOSED
{ ip: string, time: string, user: string, verb: string, `path`: string, stat: int32, size: int32 };
CREATE EXTERNAL DATASET AccessLog(AccessLogType) USING localfs
{"path":"localhost:///Users/mikejcarey/extdemo/accesses.txt"},
("format"="delimited-text"), ("delimiter"="|"));

CREATE FEED myMsgFeed USING socket_adapter
{"sockets"="127.0.0.1:10001"}, ("address-type"="IP"),
("type-name"="GleambookMessageType"), ("format"="adm"));
CONNECT FEED myMsgFeed TO DATASET GleambookMessages;
START FEED myMsgFeed;
ASTERIX Queries (SQL++ or AQL)

• **Q1**: List the user names and messages sent by Gleambook social network users with less than 3 friends:

```sql
SELECT user.name AS uname,
    (SELECT VALUE msg.message
     FROM GleambookMessages msg
     WHERE msg.authorId = user.id) AS messages
FROM GleambookUsers user
WHERE COLL_COUNT(user.friendIds) < 3;
```

```json
{
  "uname": "NilaMilliron", "messages": []
}
{
  "uname": "WoodrowNehling", "messages": [" love acast its 3G is good:" ]
}
{
  "uname": "IsbelDull", "messages": [" like product-y the plan is amazing", " like product-z its platform is mind-blowing" ]
}
...
Q2: Identify active users (last 30 days) and group and count them by their numbers of friends:

WITH endTime AS current_datetime(),
     startTime AS endTime - duration("P30D")
SELECT nf AS numFriends, COUNT(user) AS activeUsers
FROM GleambookUsers user
LET nf = COLL_COUNT(user.friendIds)
WHERE SOME logrec IN AccessLog SATISFIES
    user.alias = logrec.user
    AND datetime(logrec.time) >= startTime
    AND datetime(logrec.time) <= endTime
GROUP BY nf;

SQL++ highlights:
• Many features (see docs)
• Spatial & text predicates
• Set-similarity matching
Updates and Transactions

- **Q3:** Add a new user to Gleambook.com:

```sql
UPSERT INTO GleambookUsers (  
"id":667,"alias":"dfrump",  
"name":"DonaldFrump",  
"nickname":"Frumpkin",  
"userSince":datetime("2017-01-01T00:00:00"),  
"friendIds":{},  
"employment":[{"organizationName":"USA",  
"startDate":date("2017-01-20")}],  
"gender":"M"}
);
```

- Key-value store-like transactions (w/record-level atomicity)
- Insert, delete, and upsert ops; index-consistent
- 2PL concurrency
- WAL no-steal, no-force with LSM shadowing
Example AsterixDB Use Cases

- Potential use case areas include
  - Behavioral science
  - Cell phone event analytics
  - Social data analytics
  - Public health
  - Cluster management log analytics
  - Power usage monitoring
  - IoT data storage and querying
  - ....
Project Status

• 4 year initial NSF project (250+ KLOC), started 2009
• Now officially *Apache AsterixDB*...
  • Semistructured “NoSQL” style data model
  • Declarative queries, inserts, deletes, upserts (SQL++)
  • Scalable parallel query execution
  • Data storage/indexing (primary & secondary, LSM-based)
  • Internal and external datasets both supported
  • Rich set of data types (including text, time, location)
  • Fuzzy and spatial query processing
  • NoSQL-like transactions (for inserts/deletes)
  • Data feeds and indexes for external datasets
  • ....
Commercial Use: Big Data Analytics

Couchbase Data Platform

- Service-Centric Clustered Data System
- Multi-process Architecture
- Dynamic Distribution of Facilities
- Cluster Map Distribution
- Automatic Failover
- Enterprise Monitoring/Management
- Security
- Offline Mobile Data Integration
- Streaming REST API
- SQL-like Query Engine for JSON
- Clustered* Global Indexes
- Lowest Latency Key-Value API
- Active-Active Inter-DC Replication
- Local Aggregate Indexes
- Full-Text Search*

✓ Operational Analytics (currently DP)
For More Information

• Asterix project UCI/UCR research home
  • http://asterix.ics.uci.edu/

• Apache AsterixDB home
  • http://asterixdb.apache.org/

• SQL++ Primer
  • http://asterixdb.apache.org/docs/0.9.2/index.html