AsterixDB

Phrida Norrhall og Frida Schmidt-Hanssen
Big Data Management Systems

- Data is everywhere
- Ingest, manage, index, query, and analyze
- Three areas:
  - Semi-structured data management
  - Parallel databases
  - First-generation Big Data platforms
One size fits a bunch

1. A flexible, semi-structured data model
2. Full query language with at least the power of SQL
3. An efficient parallel query runtime
4. Data management and automatic indexing
5. Wide range of query sizes
6. Continuous data ingestion
7. Scale gracefully
8. Support for common “Big Data”-types
System Overview
Data Definitions

- **Dataverse**
  - “Data universe”

- **Datatypes**
  - Specifies what its definer wants AsterixDB to know a priori
  - Open vs. closed

- **Datasets**
  - Stored collection of data instances of a datatype
  - Can define multiple datasets of a datatype
Dataverse and Datatypes

```sql
drop dataverse TinySocial if exists;
create dataverse TinySocial;
use dataverse TinySocial;

create type EmploymentType as open {
    organization-name: string,
    start-date: date,
    end-date: date
}

create type MugshotUserType as {
    id: int32,
    alias: string,
    name: string,
    user-since: datetime,
    address: {
        street: string,
        city: string,
        state: string,
        zip: string,
        country: string
    },
    friend-ids: {{ int32 }},
    employment: [EmploymentType]
}

create type MugshotMessageType as closed {
    message-id: int32,
    author-id: int32,
    timestamp: datetime,
    in-response-to: int327,
    sender-locaton: point?,
    tags: [{ string }],
    message: string
}
```
External Data

- Supports direct access to externally resident data
  - Limited to being read-only and static
- No pre-loading

```java
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"="{hostname}://{path}",
    "format"="delimited-text",
    "delimiter"="|"));
```
Data Feeds

- Built-in mechanism for new data to continuously be ingested from external sources
- Need to index “fast flowing” data
- User queries work against the stored data, not the incoming stream

```plaintext
use dataverse TinySocial;
create feed socket_feed
    using socket_adaptor
    ("sockets"="{address}:{port}",
    "addressType"="IP",
    "type-name"="MugshotMessageType",
    "format"="adm");
connect feed socket_feed to dataset MugshotMessages;
```
AQL (Asterix Query Language)

- Based loosely on XQuery
- Queries based on FLWOR (for-let-where-order by-return)

<table>
<thead>
<tr>
<th>AQL</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>for</td>
<td>from</td>
</tr>
<tr>
<td>return</td>
<td>select</td>
</tr>
<tr>
<td>let</td>
<td>with</td>
</tr>
<tr>
<td>where</td>
<td>where</td>
</tr>
<tr>
<td>order by</td>
<td>order by</td>
</tr>
</tbody>
</table>
AQL Query Example

for $user in dataset MugshotUsers
for $message in dataset MugshotMessages
  where $message.author-id = $user.id
    and $user.user-since >= datetime('2010-07-22T00:00:00')
    and $user.user-since <= datetime('2012-07-29T23:59:59')
  return {
    "uname" : $user.name,
    "message" : $message.message
  }
System Architecture
Asterix Software Stack

- AQL
- HiveQL
- XQuery
- Algebricks Algebra Layer
- Pregel Job
- M/R Layer
- Hyracks Data-Parallel Platform
- Hyracks Job
Algebricks

- AsterixDB compiles an AQL query into an **Algebricks** algebraic program
- Program is optimized
- Translate physical query plan into corresponding Hyracks job
Hyracks

- Bottom-most layer of Asterix software stack
- Runtime layer aka. executor
- Manages data-parallel tasks requested by a higher level in the stack or direct users of Hyracks
- Jobs in form of DAGs (directed acyclic graphs)
  - Operators and connectors
Hyracks job

```sql
aggregate $agg := global-avg($lagg)
   n:1 replicating

aggregate $lagg := local-avg($l)
   1:1

assign $l := string-length($m.message)
   1:1

select $t >= 2014-01-01T00:00:00 and $t < 2014-04-01T00:00:00
   1:1

assign $t := $m.timestamp
   1:1

btree $m := search(MugshotMessages, $id, $id)
   1:1

sort $id
   1:1

btree $id := search(msTimestampIdx, $lo, $hi)
   1:1

assign $hi := 2014-04-01T00:00:00
assign $lo := 2014-01-01T00:00:00
```

avg(

for $m in dataset MugshotMessages
where $m.timestamp >= datetime("2014-01-01T00:00:00")
   and $m.timestamp <  datetime("2014-04-01T00:00:00")
return string-length($m.message)

)
Storage and Indexing

- **Log-Structured Merge (LSM) trees**
  - Underlying technology for all of its internal data storage and indexing
- **Entries initially stored in memory**
- **When exceeding specified threshold, entries are flushed to disk**
  - Avoid costly random disk I/O and support high ingestion rates
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Asterix</th>
<th>Asterix</th>
<th>Syst-X</th>
<th>Hive</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rec Lookup</strong></td>
<td>0.03</td>
<td>0.03</td>
<td>0.12</td>
<td>(379.11)</td>
<td>0.02</td>
</tr>
<tr>
<td><strong>Range Scan</strong></td>
<td>79.47</td>
<td>148.15</td>
<td>148.33</td>
<td>11717.18</td>
<td>175.84</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.10</td>
<td>0.10</td>
<td>4.90</td>
<td>(11717.18)</td>
<td>0.05</td>
</tr>
<tr>
<td><strong>Sel-Join (Sm)</strong></td>
<td>78.03</td>
<td>96.76</td>
<td>55.01</td>
<td>333.56</td>
<td>66.46</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.51</td>
<td>0.55</td>
<td>2.13</td>
<td>(333.56)</td>
<td>0.62</td>
</tr>
<tr>
<td><strong>Sel-Join (Lg)</strong></td>
<td>79.62</td>
<td>99.73</td>
<td>56.65</td>
<td>350.92</td>
<td>273.52</td>
</tr>
<tr>
<td>— with IX</td>
<td>2.24</td>
<td>2.32</td>
<td>10.59</td>
<td>(350.92)</td>
<td>14.97</td>
</tr>
<tr>
<td><strong>Sel2-Join (Sm)</strong></td>
<td>79.06</td>
<td>97.82</td>
<td>55.81</td>
<td>340.02</td>
<td>66.45</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.50</td>
<td>0.52</td>
<td>2.62</td>
<td>(340.02)</td>
<td>0.61</td>
</tr>
<tr>
<td><strong>Sel2-Join (Lg)</strong></td>
<td>80.18</td>
<td>101.24</td>
<td>56.10</td>
<td>394.11</td>
<td>313.17</td>
</tr>
<tr>
<td>— with IX</td>
<td>2.32</td>
<td>2.32</td>
<td>10.70</td>
<td>(394.11)</td>
<td>15.28</td>
</tr>
<tr>
<td><strong>Agg (Sm)</strong></td>
<td>128.66</td>
<td>232.30</td>
<td>130.64</td>
<td>83.18</td>
<td>400.97</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.16</td>
<td>0.17</td>
<td>0.14</td>
<td>(83.18)</td>
<td>0.19</td>
</tr>
<tr>
<td><strong>Agg (Lg)</strong></td>
<td>128.71</td>
<td>232.41</td>
<td>132.19</td>
<td>94.11</td>
<td>401</td>
</tr>
<tr>
<td>— with IX</td>
<td>5.53</td>
<td>5.55</td>
<td>4.67</td>
<td>(94.11)</td>
<td>8.34</td>
</tr>
<tr>
<td><strong>Grp-Aggr (Sm)</strong></td>
<td>130.20</td>
<td>232.77</td>
<td>131.18</td>
<td>127.85</td>
<td>398.27</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.45</td>
<td>0.46</td>
<td>0.17</td>
<td>(127.85)</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>Grp-Aggr (Lg)</strong></td>
<td>130.62</td>
<td>234.10</td>
<td>133.02</td>
<td>140.21</td>
<td>400.10</td>
</tr>
<tr>
<td>— with IX</td>
<td>5.96</td>
<td>5.91</td>
<td>4.72</td>
<td>(140.21)</td>
<td>9.03</td>
</tr>
</tbody>
</table>

### Table 2: Dataset sizes (in GB)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Asterix (Schema)</th>
<th>Asterix (KeyOnly)</th>
<th>Syst-X</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.091</td>
<td>0.093</td>
<td>0.040</td>
<td>0.035</td>
</tr>
<tr>
<td>20</td>
<td>0.010</td>
<td>0.011</td>
<td>0.026</td>
<td>0.024</td>
</tr>
</tbody>
</table>

### Table 4: Average insert time per record (in sec)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Asterix (Schema)</th>
<th>Asterix (KeyOnly)</th>
<th>Syst-X</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192</td>
<td>120</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>360</td>
<td>240</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>290</td>
<td>100</td>
<td>495</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>38</td>
<td>12</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>240</td>
<td>215</td>
<td>478</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- One size fits a bunch
- Full-function BDMS
- Cross between a Big Data analysis platform, a parallel RDBMS, and a NoSQL store - But different from each of them
- Offers native data storage and indexing in addition to querying of datasets
- Open data model that handles complex nested data as well as flat data
- Full query language (AQL) that supports declarative querying over multiple data sets