**GLADE: Big Data Analytics Made Easy**

Yu Cheng, Chengjie Qin, Florin Rusu
Electrical Engineering and Computer Science, University of California, Merced
ycheng4@ucmerced.edu, cqin3@ucmerced.edu, frusu@ucmerced.edu

---

### Architecture

- **GLADE** is a scalable and efficient parallel framework for Big Data analytics
- Generalized Linear Aggregates (GLA) UDA interface: **Init**, **Accumulate**, **Merge**, and **Terminate**
  - Users implement any aggregate computation as GLA

### GLADE GLAs

- **Average**: sum; count
  - `Accumulate(Tuple)` – add tuple value to sum; increment count
  - `Merge(GLA)` – add sums and counts
  - `Terminate()` – calculate average as sum divided by count
- **Group By**: hash table (user → revenue)
  - `Accumulate(Tuple)` – if find user, add revenue; otherwise insert new user
  - `Merge(GLA)` – merge hash tables; for same user, sum the revenues
- **Top-K**: min-heap with K entries
  - `Accumulate(Tuple)` – if heap top smaller than tuple, extract top and insert new tuple; reorganize heap
  - `Merge(GLA)` – call Accumulate for each element in GLA argument
- **K-Means**: K centers; K Average GLAs
  - `Accumulate(Tuple)` – find closest center and call Accumulate on corresponding Average GLA
  - `Merge(GLA)` – call Merge on corresponding Average GLAs
  - `Terminate()` – compute new centers

### PostgreSQL UDA

- **SQL definition**
  ```sql
  CREATE AGGREGATE kmeans (double precision[], double precision[])
  (STYPE = double precision[],
   SFUNC = kmeans_transit,
   FINALFUNC = kmeans_final,
   INITCOND = '(0)' );
  ```
- **C implementation**
  - **Write transition and final function of UDA in C code**
  - **Build C code as shared library**
  - **Create table as UDT to keep states between multiple iterations**
  - **Create UDA using UDFs**

### Experimental Results

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time (seconds)</th>
<th>GLADE</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>3</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>Group By</td>
<td>34</td>
<td>5,895</td>
<td></td>
</tr>
<tr>
<td>Top-K</td>
<td>5</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>K-Means (one iteration)</td>
<td>4</td>
<td>62</td>
<td></td>
</tr>
</tbody>
</table>

### Single node, 20GB

- **Data & Tasks**
  - **Average**: average time a user spends on a page
  - **Group By**: ad revenue generated by a user across all the visited web pages
  - **Top-K**: users who generated the largest one hundred ad revenues on a single visit
  - **K-Means**: five most representative ad revenues

### Hadoop Map-Reduce

- **Java & Pig Latin**
  - **Average and Top-K**
  - **Group By**
  - **K-Means**

- **Reduce**
  - Calculate local result and emit with single pre-defined key in close.

- **K-Means**
  - Map-Reduce direct application.
  - Python as external driver to control iterations. Separate key for each center.

---

Yu Cheng, Chengjie Qin, Florin Rusu (EECS, University of California, Merced)

2012 ACM SIGMOD International Conference on Management of Data