Pregelix: Big(ger) Graph Analytics on A Dataflow Engine

Yingyi Bu (UC Irvine)

Joint work with:

Vinayak Borkar (UC Irvine),
Michael J. Carey (UC Irvine),
Tyson Condie (UCLA),
Jianfeng Jia (UC Irvine)

http://isg.ics.uci.edu
Introduction

Big Graphs are becoming common

- web graph
- social network
- ......
Introduction

- How Big are Big Graphs?
  - Web: 8.53 Billion pages in 2012
  - Facebook active users: 1.01 Billion
  - de Bruijn graph: 3 Billion nodes
  - ...... 

- Weapons for mining Big Graphs
  - Pregel (Google)
  - Giraph (Facebook, LinkedIn, Twitter, etc.)
  - Distributed GraphLab (CMU)
  - GraphX (Berkeley)
Programming Model

- Think like a vertex
  - receive messages
  - update states
  - send messages
public abstract class Vertex<I extends WritableComparable, V extends Writable, E extends Writable, M extends Writable> implements Writable{

    public abstract void compute(Iterator<M> incomingMessages);

    .......
}

● Helper methods
  ○ sendMsg(I vertexId, M msg)
  ○ voteToHalt()
  ○ getSuperstep()
More APIs

● **Message Combiner**
  ○ Combine messages
  ○ Reduce network traffic

● **Global Aggregator**
  ○ Aggregate statistics over all live vertices
  ○ Done for each iteration

● **Graph Mutations**
  ○ Add vertex
  ○ Delete vertex
  ○ A conflict resolution function
Pregel Semantics

- **Bulk-synchronous**
  - A global barrier between iterations

- **Compute invocation**
  - Once per active vertex in each superstep
  - A halted vertex is activated when receiving messages

- **Global halting**
  - Each vertex is halted
  - No messages are in flight

- **Graph mutations**
  - Partial ordering of operations
  - User-defined *resolve* function
Process-centric runtime

Worker-1

Worker-2

Master

Vertex { id: 2
  halt: false
  value: 2.0
  edges: (3,1.0),
  (4,1.0)
}

Vertex { id: 4
  halt: false
  value: 1.0
  edges: (1,1.0)
}

Vertex { id: 3
  halt: false
  value: 3.0
  edges: (2,1.0),
  (4,1.0)
}

Vertex { id: 1
  halt: false
  value: 3.0
  edges: (3,1.0),
  (4,1.0)
}

Message <id, payload>

Control signal
Issues and Opportunities

- Out-of-core support

“I’m trying to run the sample connected components algorithm on a large data set on a cluster, but I get a “java.lang.OutOfMemoryError: Java heap space” error.”

26 similar threads on Giraph-users mailing list during the past year!
Issues and Opportunities

● Physical flexibility
  ○ PageRank, SSSP, CC, Triangle Counting
  ○ Web graph, social network, RDF graph
  ○ 8 machine school cluster, 200 machine Facebook data center

One-size fits-all?
Issues and Opportunities

- Software simplicity

- Pregel
- GraphLab
- Giraph
- Hama

- Vertex/map/msg data structures
- Task scheduling
- Memory management
- Message delivery
- Network management
The Pregelix Approach

<table>
<thead>
<tr>
<th>Relation</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
<td>(vid, halt, value, edges)</td>
</tr>
<tr>
<td>Msg</td>
<td>(vid, payload)</td>
</tr>
<tr>
<td>GS</td>
<td>(halt, aggregate, superstep)</td>
</tr>
</tbody>
</table>
Pregel UDFs

- **compute**
  - Executed at each active vertex in each superstep

- **combine**
  - Aggregation function for messages

- **aggregate**
  - Aggregate function for the global states

- **resolve**
  - Used to resolve graph mutations
Logical Plan

\[ \gamma_{vid} \text{ combine} \]

\[ \sigma(V.halt = \text{false} \lor M.payload \neq \text{NULL}) \]

D2
D3
D4, D5, D6
D7

<table>
<thead>
<tr>
<th>Flow</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>Vertex tuples</td>
</tr>
<tr>
<td>D3</td>
<td>Msg tuples</td>
</tr>
<tr>
<td>D7</td>
<td>Msg tuples after combination</td>
</tr>
</tbody>
</table>
Logical Plan

**Flow** | **Data**
---|---
D4 | The global halting state contribution
D5 | Values for aggregate
D8 | The global halt state
D9 | The global aggregate value
D10 | The increased superstep

**Flow** | **Data**
---|---
D6 | Vertex tuples for deletions and insertions

**Vertices**

**UDF Call** | **Data**
---|---
D1, D2, D3, D4, D5 | 

**GS**

**UDF Call** | **Data**
---|---
D1, D2, D3, D6 | 

**GS** | **Data**
---|---
D8, D9 | 

**GS** | **Data**
---|---
D10 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 | 

**GS**

**Superstep** | **Data**
---|---
G.superstep+1 |
The Pregelix System

Pregel Physical Plans

Vertex/map/msg data structures
Task scheduling
Memory management
Message delivery
Network management

Operators
Access methods
Task scheduling
Record/Index management
Data exchanging
Buffer management
Connection management

A general purpose parallel dataflow engine
The Runtime

- Runtime Choice?
  - Hyracks
  - Hadoop

- The Hyracks data-parallel execution engine
  - Out-of-core operators
  - Connectors
  - Access methods
  - User-configurable task scheduling
  - Extensibility
Parallelism

Worker-1

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
<th>halt</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.0</td>
<td>false</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
<td>false</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
<th>halt</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
<th>halt</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>false</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>false</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
<th>halt</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Worker-2

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>vid</th>
<th>msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```
Physical Choices

- **Vertex storage**
  - B-Tree
  - LSM B-Tree

- **Group-by**
  - Pre-clustered group-by
  - Sort-based group-by
  - HashSort group-by

- **Data redistribution**
  - m-to-n merging partitioning connector
  - m-to-n partitioning connector

- **Join**
  - Index Full outer join
  - Index Left outer join
Data Storage

- **Vertex**
  - Partitioned B-tree or LSM B-tree
- **Msg**
  - Partitioned local files, sorted
- **GS**
  - Stored on HDFS
  - Cached in each worker
Physical Plan: Message Combination

- **Sort-Groupby-M-to-N-Partitioning**
- **HashSort-Groupby-M-to-N-Partitioning**
- **Sort-Groupby-M-to-N-Merge-Partitioning**
- **HashSort-Groupby-M-to-N-Merge-Partitioning**

M-to-N Partitioning Connector

M-To-N Partitioning Merging Connector
Physical Plan: Message Delivery

\[ \sigma (V.halt = false \ || \ M.paylod \neq NULL) \]

Index Full Outer Join

\[ M.vid = V.vid \]

\[ \sigma (halt = false) \]

Function Call (NullMsg)

\[ D1 \]

UDF Call (compute)

\[ D2 \rightarrow D6 \]

Index Left Outer Join

\[ M.vid = V.vid \]

Merge (choose())

\[ M.vid = I.vid \]

\[ M.vid = V.vid \]
Caching

Pregel, Giraph, GraphLab all have caches for this kind of iterative jobs. What do you do for caching?

- Iteration-aware (sticky) scheduling?
  - 1 Loc: location constraints

- Caching of invariant data?
  - B-tree buffer pool -- customized flushing policy: never flush dirty pages
  - File system cache -- free
Experimental Results

- Setup
  - Machines
    - a UCI cluster ~ 32 machines
      - 4 cores, 8GB memory, 2 disk drives.
  - Datasets
    - Yahoo! webmap (1,413,511,393 vertex, adjacency list, ~70GB) and its samples.
    - The Billions of Tuples Challenge dataset (172,655,479 vertices, adjacency list, ~17GB), its samples, and its scale-ups.
  - Giraph
    - Latest trunk (revision 770)
    - 4 vertex computation threads, 8GB JVM heap
Execution Time

PageRank Execution Time

PageRank Avg. Iteration Time

In-memory

Out-of-core
Execution Time

SSSP Execution Time

SSSP Avg. Iteration Time

In-memory

Out-of-core
Execution Time

CC Execution Time
- Pregelix
- Giraph-mem
- Giraph-ooc

CC Avg. Iteration Time
- Pregelix
- Giraph-mem
- Giraph-ooc
Parallel Speedup

- **Pregelix Parallel Speedup (PageRank)**
  - Relative Avg. Iteration Time vs. Number of Machines
  - Legend: X-Small, Small, Medium, Large, Ideal

- **PageRank Parallel Speedup**
  - Relative Exe. Time vs. Number of Machines
  - Legend: Pregelix, Giraph-mem, Ideal
Parallel Scale-up

![Graph showing Pregelix Parallel Scale-up](graph.png)

- **Relative Avg. Iteration Time**
- **Scale**
- **Legend**:
  - PageRank
  - SSSP
  - CC
  - Ideal
Throughput

Throughput (Always in-memory)

Throughput (In-memory to minor disk usage)

Throughput (In-memory to disk-based)

Throughput (Always disk-based)
Plan Flexibility

**PageRank Avg. Iteration Time**
- **Left Outer Join**
- **Full Outer Join**

**SSSP Avg. Iteration Time**
- **Left Outer Join**
- **Full Outer Join**

**CC Avg. Iteration Time**
- **Left Outer Join**
- **Full Outer Join**

**SSSP Avg. Iteration Time**
- **Pregelix-Left-Outer-Join**
- **Giraph**

- In-memory
- Out-of-core
- 15x
## Software Simplicity

- **Lines-of-Code**
  - Giraph: 32,197
  - Pregelix: 8,514
More systems

PageRank on Webmap-X-Small

<table>
<thead>
<tr>
<th>System</th>
<th>16nodes</th>
<th>32nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphLab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pregelix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Giraph</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution Time (sec.)
More Systems

SSSP Execution Time

- GraphLab
- Pregelix
- Giraph

Execution Time (Sec.)

- 16nodes
- 32nodes
Related Work

- **Parallel Data Management**
  - Gama, GRACE, Teradata
  - Stratosphere (TU Berlin)
  - REX (UPenn)
  - AsterixDB (UCI)

- **Big Graph Processing Systems**
  - Pregel (Google)
  - Giraph (Facebook, LinkedIn, Twitter, etc.)
  - Distributed GraphLab (CMU)
  - GraphX (Berkeley)
  - Hama (Sogou, etc.) — Too slow!
Conclusions

- Pregelix offers:
  - Transparent out-of-core support
  - Physical flexibility
  - Software simplicity

- We target Pregelix to be an open-source production system, rather than just a research prototype:
  - http://pregelix.ics.uci.edu
Q & A