Trinity

A Distributed Graph Engine on a Memory Cloud
Motivation

Polarized view on previous approaches:

<table>
<thead>
<tr>
<th></th>
<th>Graph Database</th>
<th>Query Processing</th>
<th>Graph Analytics</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neo4J</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HyperGraph DB</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MapReduce</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Pregel</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Concept

- Globally shared memory
- Distributed key/value store
- Random access abstraction for online queries (BFS, graph-matching)
- Scalability through partitioning
- Restrictions on message passing for improved performance
Architecture 1/2

Client Lib

Trinity Proxy

Trinity Slaves

Client Lib

Client Lib

Lib Client

Lib Client

Trinity Proxy
Architecture 2/2

Graph Operations
GetInlinks(), Outlinks.Foreach(...), etc

Graph Model

Trinity Specification Language

Memory Cloud
(Distributed Key-Value Store)

Distributed Memory Storage
Message Passing Framework
```csharp
// TSL Script

cell struct MyCell
{
    int Id;
    List<long> Links;
}

// Generated API

using(var cell = UseMyCellAccessor(cellId))
{
    int Id = cell.Id;  // Get the value of Id
    cell.Links[1] = 2;  // Set Links[1] to 2
}

// Blob View

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000011</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000010</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000011</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>
```

Cell Schema Defined in TSL

Compile

Cell Accessor

Blob

Manipulate MyCell via Cell Accessor
Message passing model

- Having all messages in memory: infeasible
- Assumption: fixed set of messaging partners
- Bipartite view effective but costly, better:

```
+-----------------+-----------------+
| UID             | UID             |
| neighbors       | message         |
| attributes      |                 |
| local variables |                 |
| message         |                 |
+-----------------+-----------------+
```

Type A vertices

Type B vertices
Trinity Specification Language

Object oriented manipulation through cell accessor abstraction

```csharp
[CellType: NodeCell]
cell struct Movie
{
    string Name;
    [EdgeType: SimpleEdge, ReferencedCell: Actor]
    List<long> Actors;
}
```
Evaluation - BFS

(a) PBGL

Execution Time (sec)

Node Count (Million)

(b) Trinity

Execution Time (sec)

Node Count (Million)
Evaluation – Memory usage

(c) PBGL

(d) Trinity
Conclusion

- Unified approach for distributed graph processing
- Efficient shared memory abstraction
- Fault recovery through checkpointing (depending on task)
- Library / coordination unclear
- No ACID transactions