Trinity: A Distributed Graph Engine on a Memory Cloud

By Shao et al.
Presented by Ruoxi Zhang
Reference

### Online Query Processing

- Requires low latency
- Graph exploration
  - Random access
- e.g., BFS, subgraph matching, etc.

### Offline Graph Analytics

- Requires high throughput
- Iterative, batch processing
- Parallel computing
- e.g., PageRank, betweenness computation, etc.
Why Trinity?

Characteristics of graph computation
- Fast random data access

Improve performance
- Keeping data in main memory

The scale of data
- Distributed parallel computation
Trinity

- Trinity is a general purpose graph engine over a distributed memory cloud.

- **Online** query processing + **offline** graph analytics

- The belief of **all-in-memory** solutions
  - High-speed network
  - Low price DRAM

- No comprehensive built-in graph computation modules
  - Flexible data
  - Computation modeling capability
### Distributed Memory Cloud

**Trinity System Layers**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Operations</td>
<td>GetInlinks(), Outlinks.Foreach(...), etc</td>
</tr>
<tr>
<td>Graph Model</td>
<td></td>
</tr>
<tr>
<td>Trinity Specification Language</td>
<td></td>
</tr>
<tr>
<td>Memory Cloud</td>
<td>(Distributed Key-Value Store)</td>
</tr>
<tr>
<td>Distributed Memory Storage</td>
<td></td>
</tr>
<tr>
<td>Message Passing Framework</td>
<td></td>
</tr>
</tbody>
</table>

- Partition memory space into $2^p (> m)$ trunks.
  - Trunk level parallelism
  - Efficient hashing
- Basic data structure: key-value pairs
  - **Key**: 64-bit globally unique identifiers (UID)
  - **Value**: blobs of arbitrary length
  - **Metadata**: spinlock

**memory cloud → fast random access → fast graph exploration**
**Partitioning & Addressing**

- **[Scalable] Hashing mechanism**
  - Trunk number to machine ID
  - $\text{UID} \rightarrow p \ \text{bit } i \in [0, 2^p - 1]
  - Addressing table

- **[Fault-tolerant] memory trunks are backed up in TFS (Trinity File System).**

- Hash again to find the value
  - Use the hash table in trunk
  - $\text{UID} \rightarrow \text{offset} \& \text{size}
Circular Memory Management

- Given: a large number of key-value pairs; size may increase or decrease
- Goal: avoid memory gaps
- Append head
- **Defragmentation** daemon
Data Model

Trinity System Layers

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

- **Graph Model**

- **Trinity Specification Language**

- **Memory Cloud**
  - (Distributed Key-Value Store)

- **Distributed Memory Storage**
- **Message Passing Framework**

- **Cell** = value + schema
  - a node (or a rich edge)

- **TSL**
  - **Object-oriented** cell manipulation
    - Cell accessor
  - Data integration
    - Transparent query processing
    - Automatic data conversion
  - System extension
    - Models cell schema
    - Models message passing
Computation Paradigms (1)

- **Online** queries processing (e.g., traversal & subgraph matching)
  - Fast random access & parallel computing, no index

- **Offline** graph analytics (e.g., PageRank)
  - Restricted vertex-centric computation model
  - Aggregate local answers & probabilistic inference

- Higher performance and lower memory usage

<table>
<thead>
<tr>
<th>PBGL, a parallel graph processing library</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giraph, offline system</td>
<td>PageRank</td>
</tr>
</tbody>
</table>
Message passing optimization
- Create a bipartite partition of the local graph
- Buffer messages from hub vertices
- Obtain messages from vertices in other partitions on demand
- Given a data access pattern:

Vertices in a partition currently scheduled to run

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$u$</td>
</tr>
<tr>
<td></td>
<td>$v$</td>
</tr>
<tr>
<td>$y$</td>
<td></td>
</tr>
</tbody>
</table>

Computation Paradigms (2)
Fault Tolerance

- Maintain a primary replica of the shared addressing table on a leader machine

- Heartbeat messages to detect machine failures

- Fault recovery varies by computation models
  - BSP (batch synchronous processing): checkpoints
  - Asynchronous: periodical interruption
Partition

- Divide a graph into many equal size parts, such that the number of edges among them is minimized.
- Goal: load balance + reduce communication overhead
- Billion-node graph partitioning is an unsolved problem on general-purpose graph platforms.
- Multi-level partitioning algorithm
Summary

- **General purpose** – algorithms & graphs & computation models
- **Large scale** – billions of nodes
- **Distributed** – instead of storing it centrally on a single machine
- **Memory-based** – keep the graph in memory, at least the topology

Trinity is a ... graph engine.
Related Publications


- Wanyun Cui, Yanghua Xiao, Haixun Wang, Ji Hong, and Wei Wang, *Local Search of Communities* in Large Graphs, SIGMOD 2013.
Discussion

- What is unique about in-memory engine design compared to disk-based engines?

- Why wouldn’t regular disk-based databases adopt the same techniques?

- In reality, Trinity does not provide ACID transaction support. If it is added as a feature, what is the trade-off?

- What are the advantages and disadvantages of a general-purpose system?