Trinity:
A Distributed Graph Engine on a Memory Cloud

Review of:
B. Shao, H. Wang, Y. Li. Trinity: a distributed graph engine on a memory cloud, Proc. ACM SIGMOD International Conference on Management of Data, pages 505-516, 2013.

Project:
http://research.microsoft.com/en-us/projects/trinity/default.aspx

Presented by:
Jeff Avery
CS 848: Modern Database Systems
Mar 10, 2015

Outline

1. Introduction
2. Overview of Trinity
3. Architecture & Design
   - Memory cloud
   - Data model
   - Graph computation paradigms
   - Implementation
4. Experimental Evaluation
5. Conclusion

Types of Graph Queries

Offline Queries

Batch processing, iterative
High throughput

Example:
Running PageRank on a series of web pages, and displaying the results.

Online Queries

Real-time queries, random
Low latency

Example:
Finding the path between nodes in a social network (friend-of-a-friend search).

Hybrid Queries

Low latency + High throughput

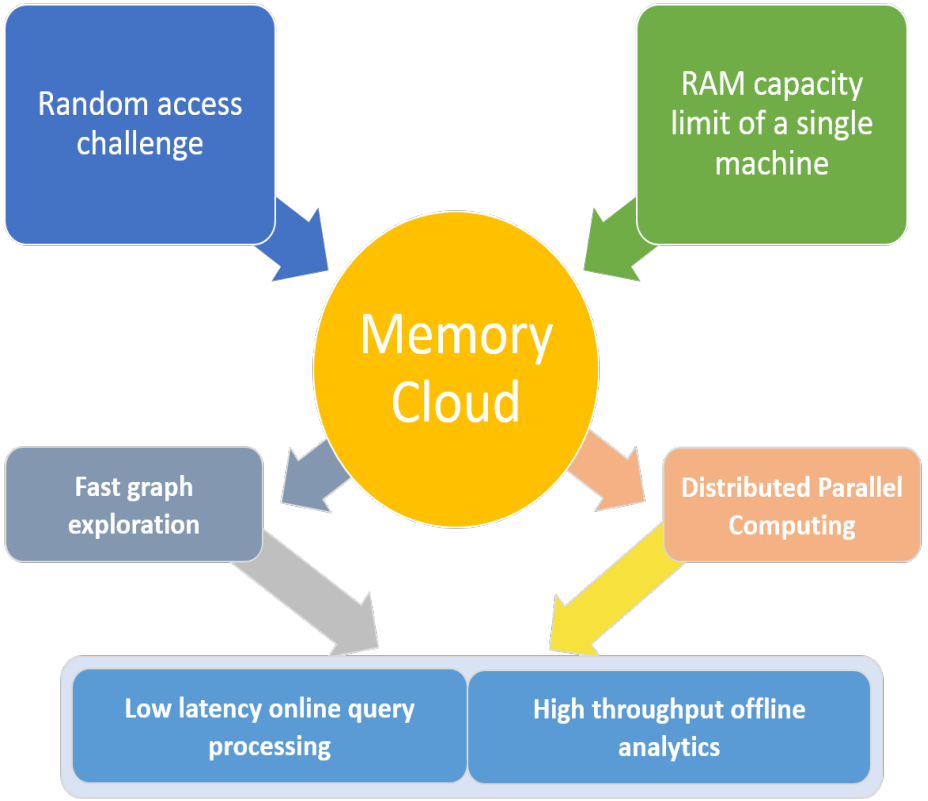
Example:
Online shortest distance calculation, using landmark nodes discovered offline.

What is Trinity?

Trinity is a “storage infrastructure and computation framework” that “organizes multiple machines into a globally addressable, distributed memory address space to support large graphs.” [Shao 2013].

Trinity provides

- Distributed computation for off-line analytics.
- Shared memory for online query processing.

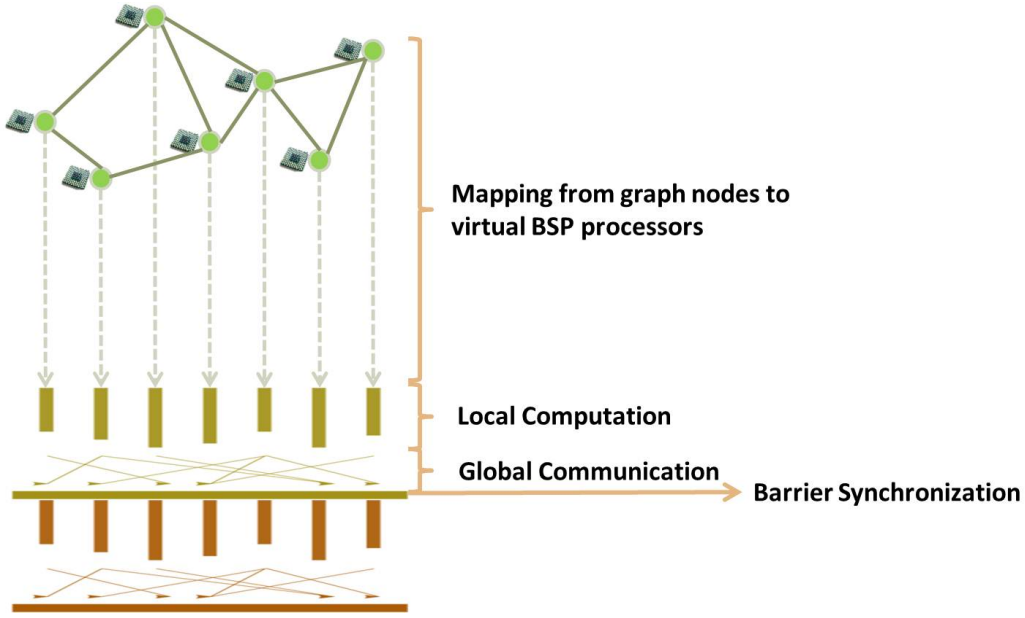


http://research.microsoft.com/en-us/projects/trinity/

Trinity Computation Model

Trinity supports a vertex-based computation model

- Similar to Pregel, based on BSP model.
- Iterative, with each step
  - Nodes send and receive messages
  - Nodes perform local computation.



Trinity's BSP Computation Model

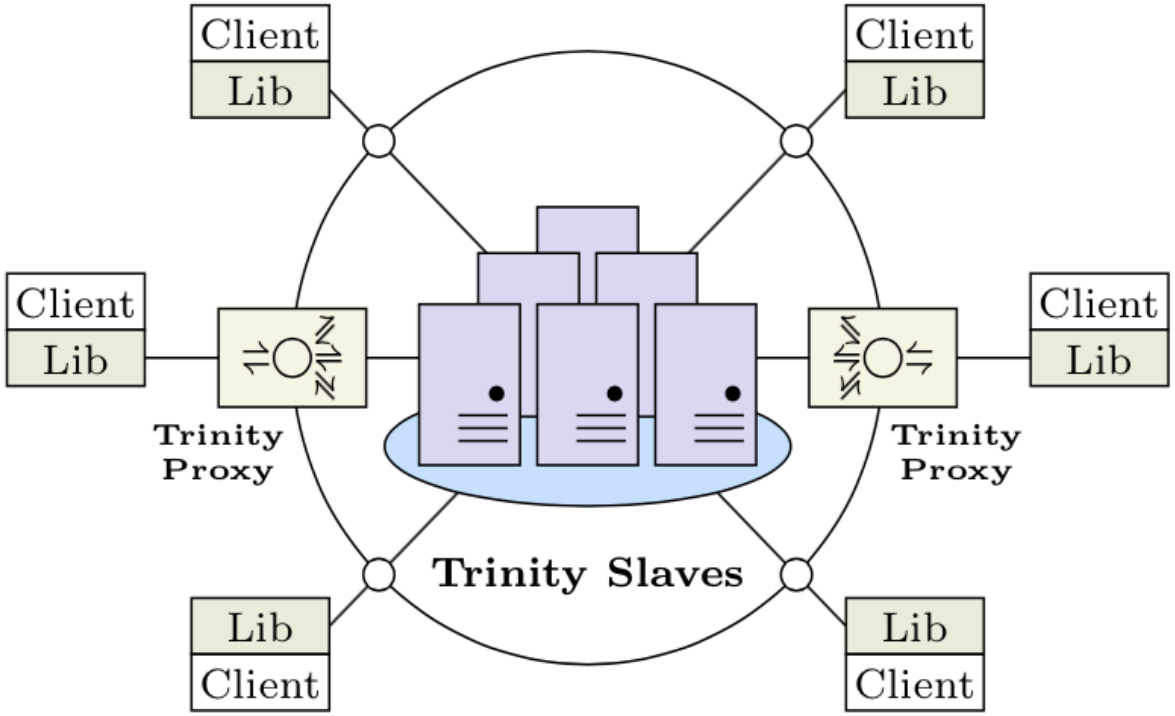
From: Trinity Project Page. Authors unknown. Retrieved 2015-03-06. http://research.microsoft.com/en-us/projects/trinity/applications.aspx.

Trinity Cluster Structure

Slave
Stores data, sends messages and computes.

Proxy (optional)
Aggregates data, forwards messages.

Client
User-interface layer.

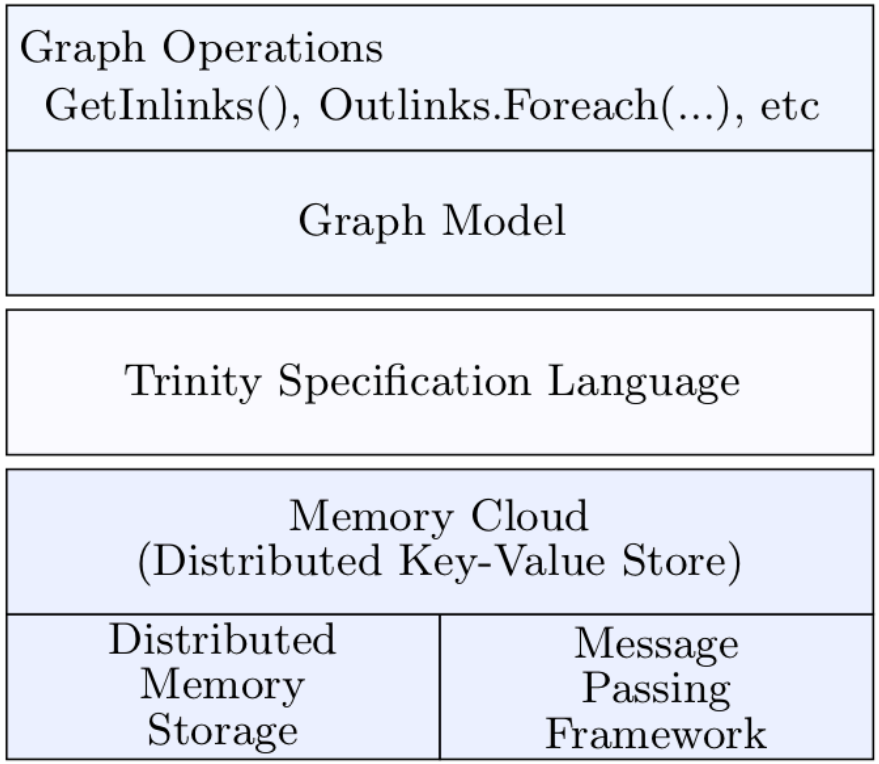


A Trinity cluster, including slaves, proxies and clients.

System Layers

Trinity is essentially a distributed key-value store.

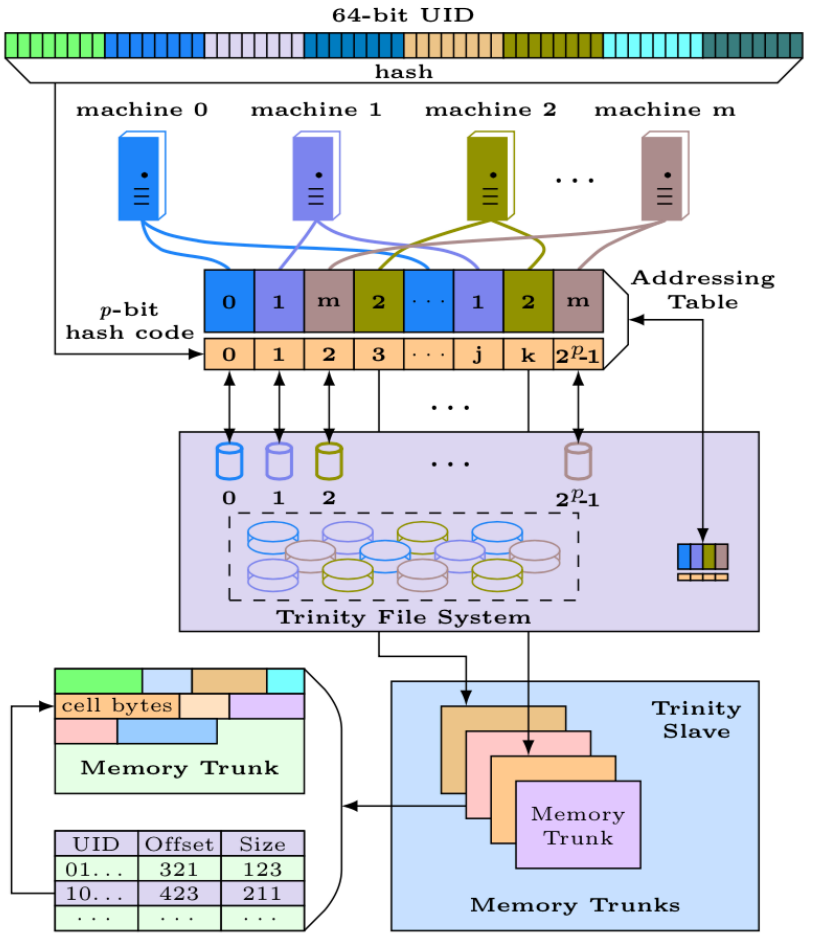
- Memory storage manages memory allocation and management, concurrency.
- Message passing supports efficient communication between nodes.
- TSL bridges the graph and data model.



System Layers

Data Partitioning & Addressing

- Key-value store: 64-bit keys, blobs.
- Memory is partitioned into memory trunks (multiple per machine).
- Hashing is used to locate a memory trunk, and an addressing table is used to locate the appropriate machine.
- Memory trunks are backed up to Trinity File System (TFS) for data-persistence.
- Machines can dynamically join/leave the memory cloud.



Data partitioning. The addressing table ties trunks to physical machines

Modeling Graphs

- Trinity is concerned with *efficient* representation of graph nodes and edges, in a way that that facilitates graph traversal.
  - Nodes are represented in the key-value store as cells with unique keys or cellIds:

$$(\mathbf{key}, \mathbf{value}) \longrightarrow (\mathbf{cellId}, \mathbf{cell})$$

    - Cells contains all of the information for that node:
      - Edges are represented by as a list of cellIDs of adjacent cells (e.g. *undirected* graph: single list, *directed* graph: incoming and outgoing lists).
      - Properties (e.g. name, type, weight).

Trinity Specification Language (TSL)

"It is hard, if not entirely impossible, to support efficient general purpose graph computation using fixed graph schema" [Shao 2013]

- TSL defines graph structure and network communication protocols.

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

Movie and Actor nodes modeled in TSL. Edges are defined as lists of Cell Ids.

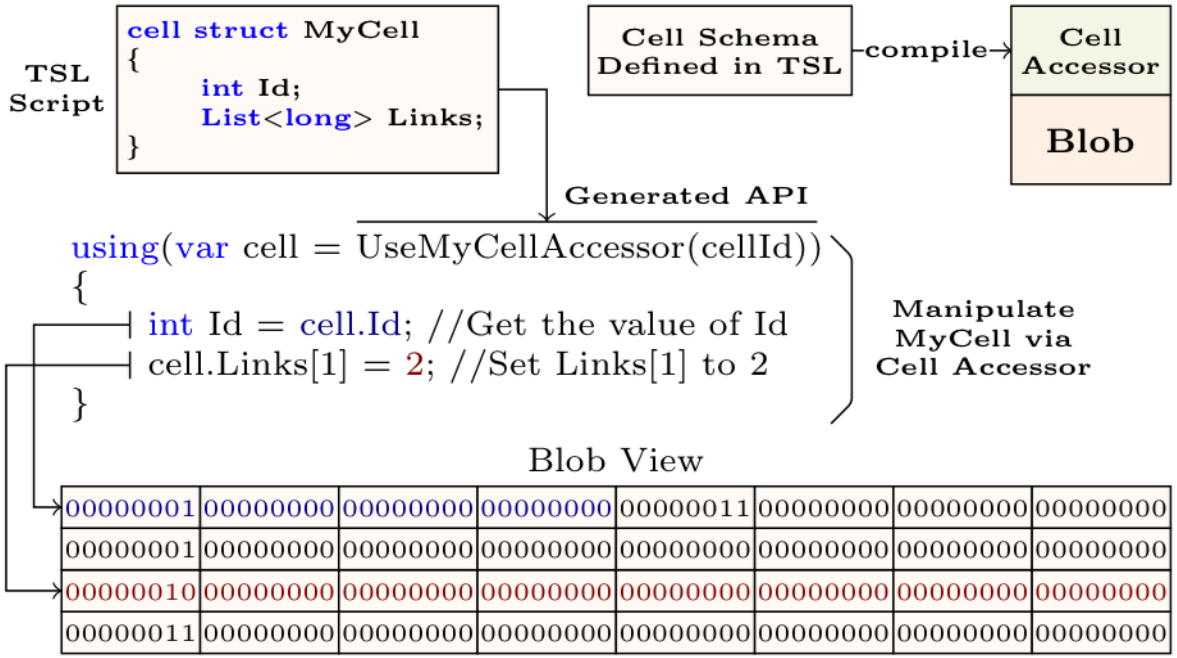
Object-Oriented Cell Manipulation

- TSL models the data, but how do we store it?

	Runtime objects	Blobs
Referencing	Cannot be referenced across machine boundaries	Assigned unique Ids and globally addressable.
Memory cost	High memory cost (e.g. empty runtime object consumes 24 bytes on the 64-bit .NET platform!)	Very low cost (stream of bytes)
Serialization to TFS	Costly to serialize and deserialize.	Cheap and fast to serialize and deserialize.

Object-Oriented Cell Manipulation

- Blobs aren't very user-friendly
  - TSL describes graph structure, exposes data mapping methods.



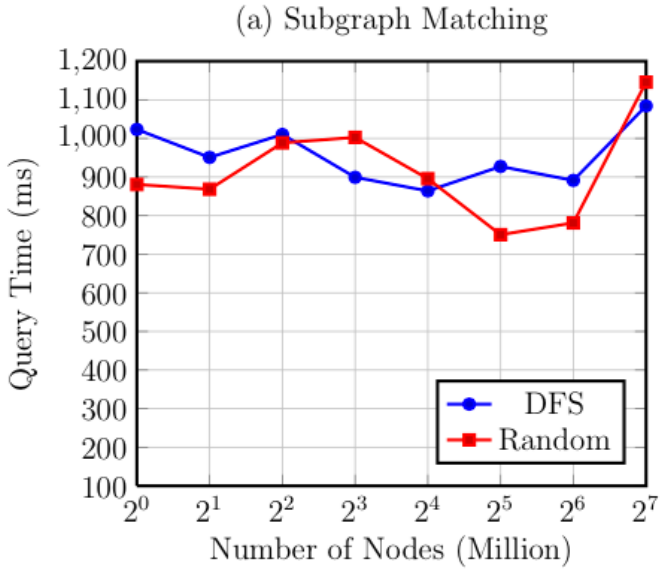
Cell access. The TSL Schema defines the data structure, which cell accessor methods expose.

Traversal Based Online Queries

- Example: On a social network, find anyone whose first name is "David" among his/her friends, his/her friends' friends and so on.
  - Searching each user by index in a large-scale web graph does not scale.
  - Trinity enables efficient BFS and DFS queries over large graph data.
    - On a "Facebook-like sized graph" consisting of 800 million nodes, 104 billion edges: exploring the entire 3-hop neighborhood on an 8-node cluster took 100 ms.

A New Paradigm for Online Queries

- "In Trinity, the combination of fast random access and parallel computing, offers a new paradigm which enables us to rethink efficient query processing on web-scale graphs" [Shao 2013].



Subgraph matching (1 million to 128 million nodes, average node degree 16, average query size 10 nodes)

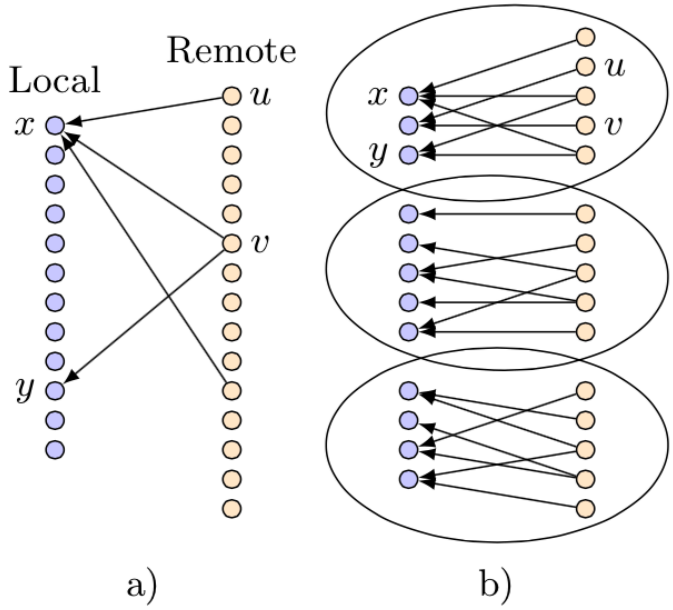
Vertex-Centric Offline Analytics

- Trinity offers a vertex-centric computation (BSP) model.
  - In each super-step:
    - A vertex receives messages, performs some computation, and sends out messages as-needed to other vertices.
    - Pregel allows messages to be sent between any nodes, while Trinity restricts messages to a subset of vertices.
  - Why does this matter?
    1. Many algorithms only require access to closest vertices (e.g. PageRank, shortest path).
    2. This allows opportunities to optimize message passing (since we have a predictable communication pattern).

Message Passing Optimization

Trinity partitions vertices (b), to reduce the number of messages that need to be sent.

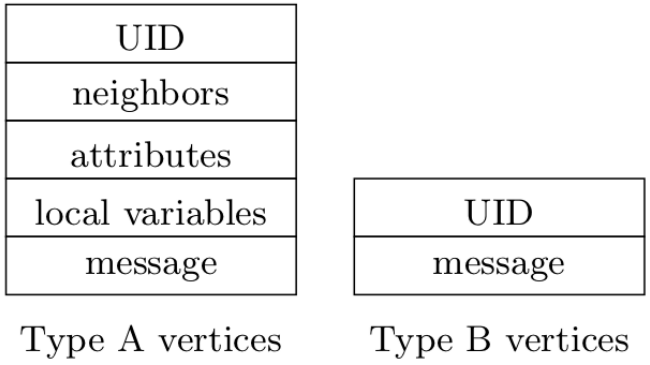
- Use *hub* vertices to store messages for the entire iteration, and just partition the remaining vertices in the neighborhood.
- Messages are grouped to avoid redundancy (*u -> x, y*)



Bipartite View on a Local Machine.
Partitioning (b) reduces message overhead.

Memory Optimization

- Observation: the data access pattern of offline analytics can be predicted.
  - This means that we don't need to store the entire graph in memory!
    - e.g. for the Facebook social-graph described, this is a savings of 708 GB memory space.
  - Less memory = fewer machines for the same performance.



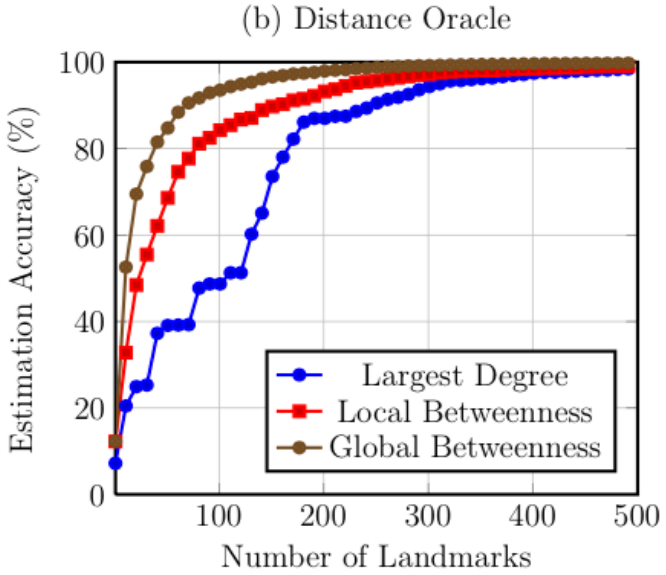
Memory-resident cell structures

Type A: Vertices in a partition currently scheduled to run on a machine.

Type B: All of the other vertices.

A New Paradigm for Offline Analytics

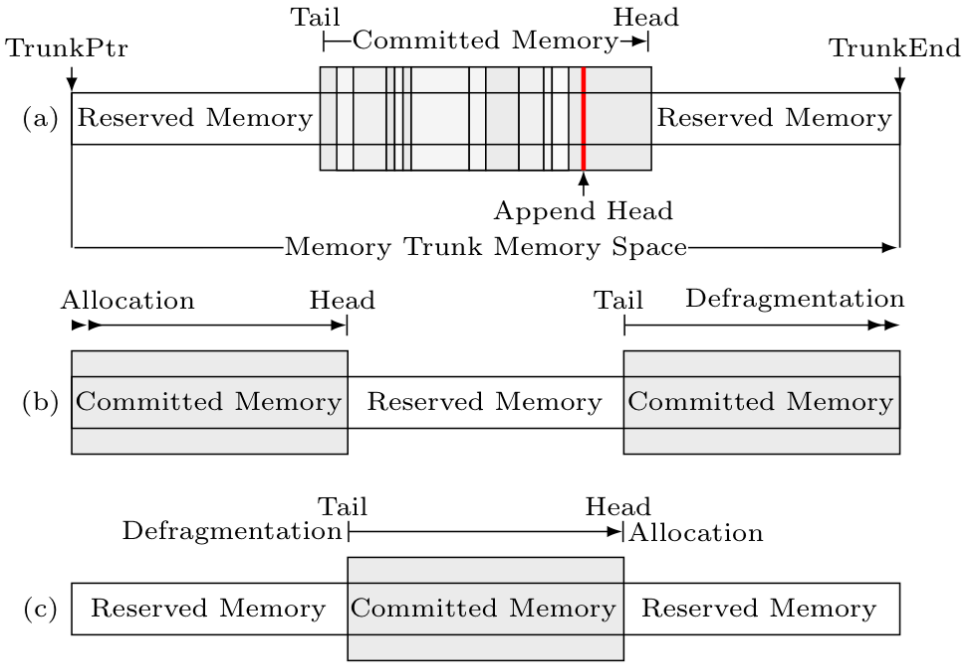
- "Can we use probabilistic inference to derive the answer for the entire graph from the answer on a single machine?" [Shao 2013]
  - e.g. 1 in 10 machines has 10% of the nodes and edges, and links directly to a large percentage of remaining nodes.
  - "Local betweenness" has good accuracy from a single vertex.



Calculating landmarks for shortest-distance calculation.

Circular Memory Management

- Goal of avoiding memory gaps when allocating memory.
- Memory allocation
  - 2 GB reserved / memory trunk
  - Allocate from the *append head;* if insufficient, allocate more and advance *committed head*.
- A defragmentation daemon moves key-value pairs towards the append head as needed.
- Memory reservation allows short-lived over-allocation and prevents aggressive defragmentation.



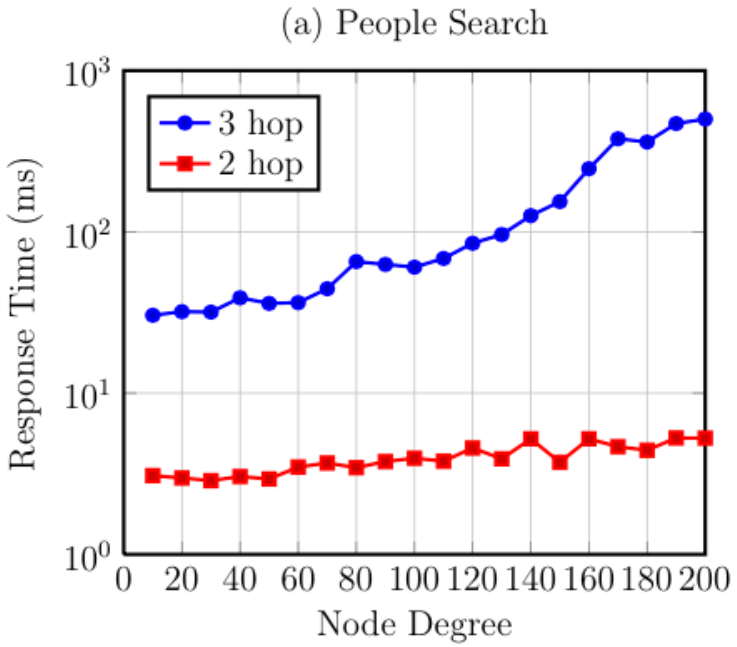
Circular Memory Management

Fault Tolerance

- Shared Address Table Maintenance
  - The addressing table (which maps machine:trunk) is a shared global data structure. Primary replica is maintained on a *leader* machine and persisted to TFS.
  - If a machine fails, the leader starts recovery (below), updates the addressing table and broadcasts the change.
  - If a leader fails, a new one is elected.
- Fault Recovery
  - BSP-based synchronous computation: checkpoints to TFS, which can be loaded by any machine over TFS for recovery.
  - Asynchronous computation: snapshots.

Trinity: Graph Traversal

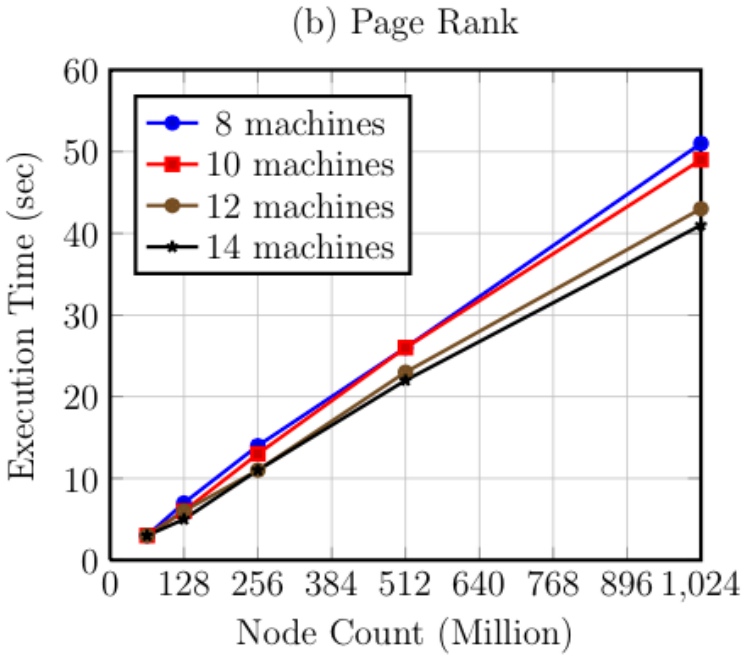
- People-search is an example of an online, traversal-based query.
- Measures query response time of searching friends by name within 2 and 3 hops on social graphs.
  - The response times of 2-hop queries are under 10 ms.
  - The response time of 3-hop search on the graph with 130 node degree (e.g. Facebook) is 96.2 ms.
- Suggests that the “David” example could be run on Facebook in under 100 ms.



Trinity People Search Query Results (8 nodes, each with 96 GB RAM, 2x 2.67 GHz Xeon Processors).

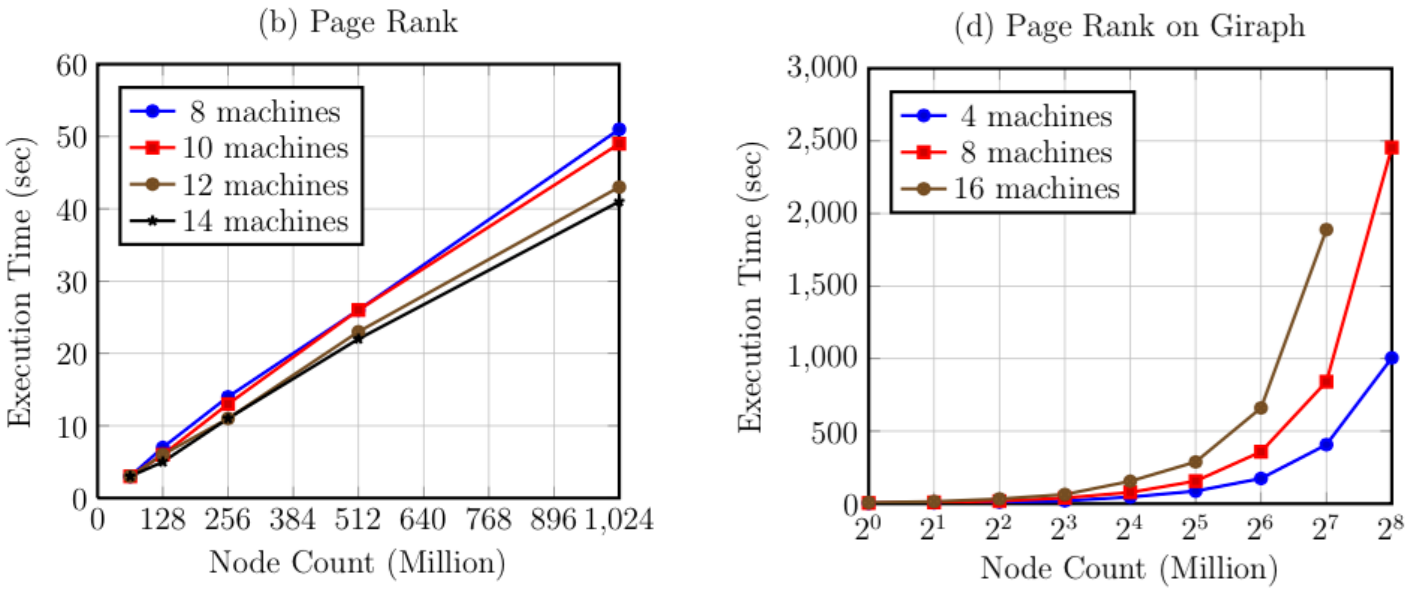
Trinity: Page Rank

- Common offline web graph analytics task.
- Measures computation time for one iteration (a super-step in BSP model) on 8, 10, 12 and 14 machines.
  - One page rank iteration on a graph with 1 billion nodes can be completed in one minute.



Page Rank Results
(8 nodes, each with 96 GB RAM, 2x 2.67 GHz Xeon Processors).

Trinity vs. Giraph: Page Rank

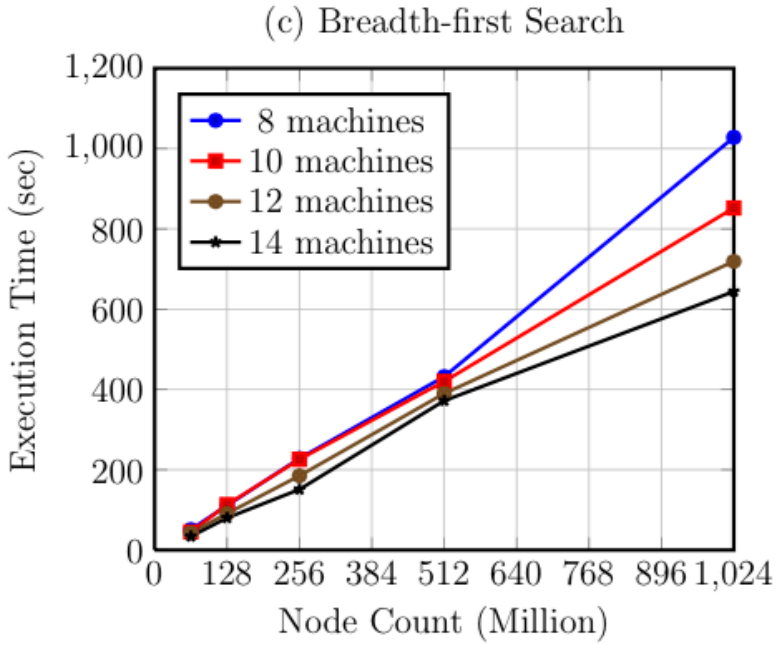


Page Rank for Trinity vs. Giraph

- Configuration: Trinity: 8 nodes, Giraph: 16 nodes (all nodes have 96 GB RAM, 2x2.67 GHz Xeon Processors).
- Results: Trinity is faster by roughly 100x. Giraph also ran out of memory in some configurations.

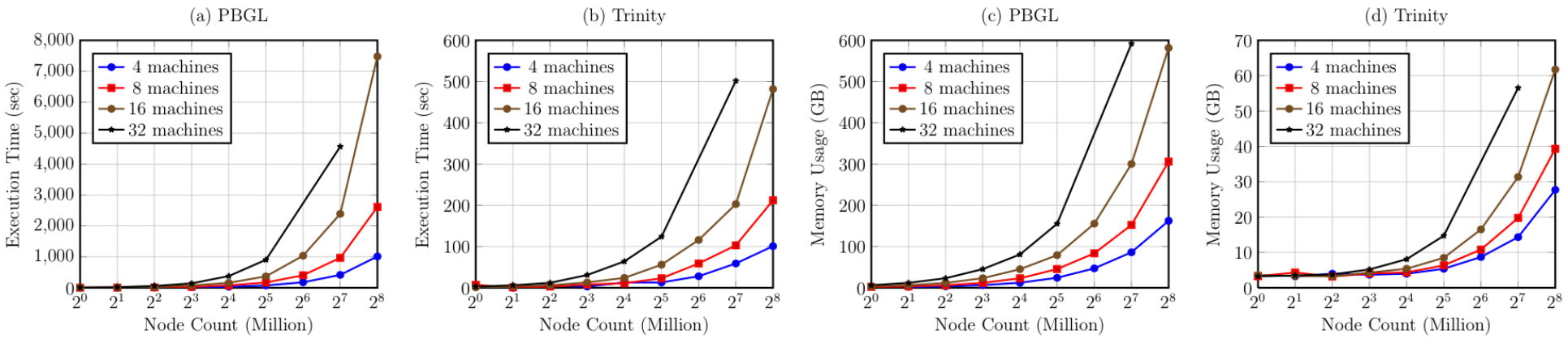
Trinity: Breadth First Search

- Breadth First Search (BFS) is a fundamental graph computation operation, often used for benchmarks.
- Measures performance of BFS on 8, 10, 12 and 14 machines.
  - For the 1 billion node graph, it takes 1028 seconds on 8 machines, and 644 seconds on 14 machines.



BFS Results
(8 nodes, each with 96 GB RAM, 2x 2.67 GHz Xeon Processors).

Trinity vs. PBGL: Breadth First Search



BFS for Trinity vs. PBGL

- Configuration: 16 machine cluster (all nodes have 96 GB RAM, 2x2.67 GHz Xeon Processors).
- Results: Trinity is faster by roughly 10x, with a 10x reduced memory footprint. PBGL also ran out of memory in some configurations.

Conclusions & Take-Away

- Trinity is a graph-engine specifically designed to address both online and offline graph analytics queries.
- It exhibits better performance than other, generic solutions, and runs efficiently on commodity hardware.
- The ability to run online queries against large graph data opens new possibilities for analyzing this data.

Questions

- Feedback?
- Comments?