Graph Databases

Chao Chen
Bryce Richards
Meng Wang
Alfred Zhong

Roadmap

- Introduction to Graph DB
- Neo4j
- GraphLab: A New Framework For Parallel Machine Learning

NoSQL Database Categories

- document-oriented databases (a.k.a document-store) store data in document.
  - MongoDB
- Key-Value stores
  - Dynamo, Cassandra
- **Graph Database**
  - Apply graph theory in the storage of information about the relationship between entries

Graph DB Model:representation

- Core Abstractions:
  - Nodes
  - Relationships between nodes (edges)
  - Properties on both

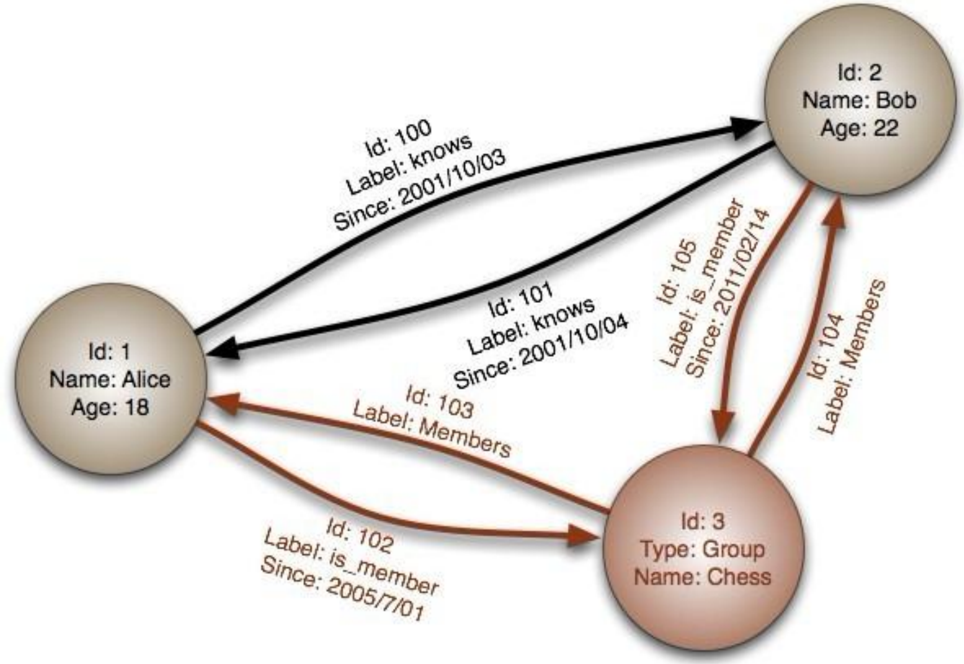
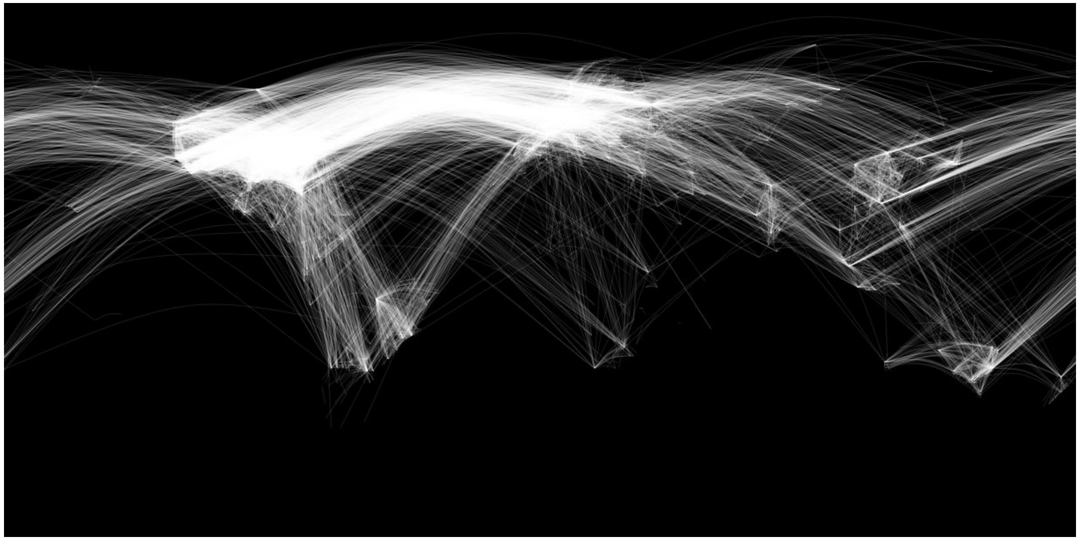
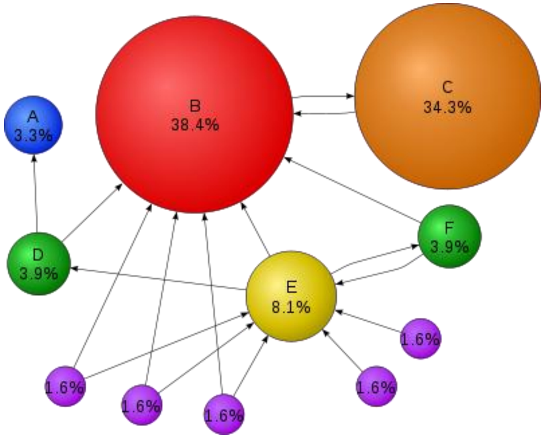


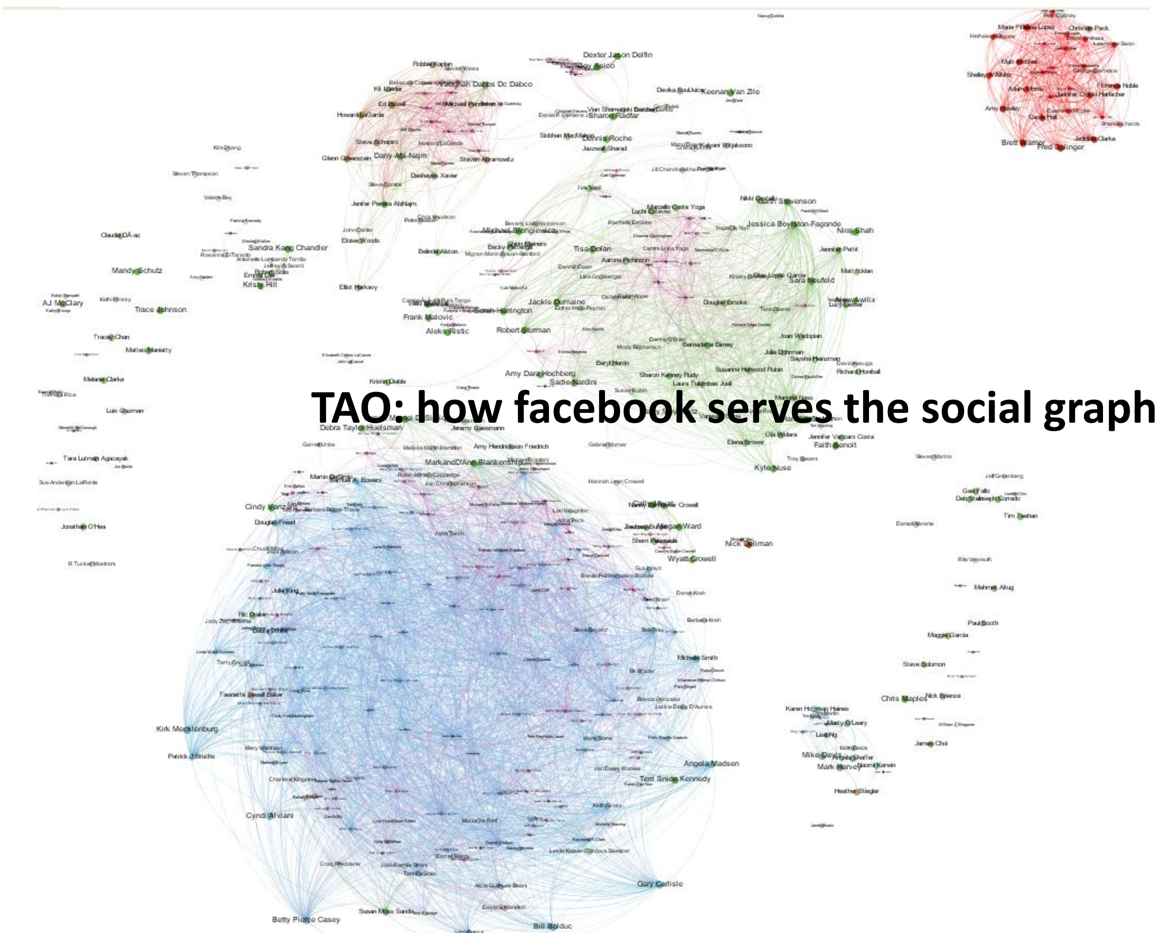
Image source: http://en.wikipedia.org/wiki/File:GraphDatabase_PropertyGraph.png

Example: Internet





Example: Social Network



TAO: how facebook serves the social graph

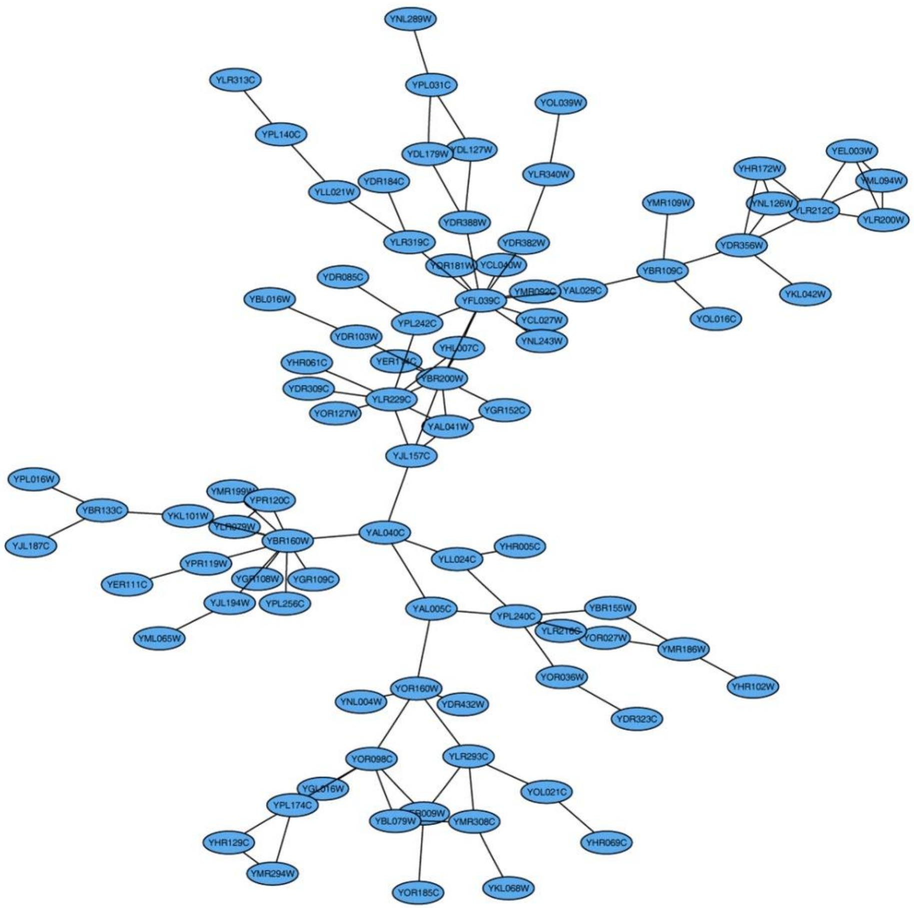
Recommendation Systems

amazon

Buy.com

YouTube

Computational Biology



Protein Interaction

“Relational database is not good for relationship data” - JOIN

- Friend links on a social network

- “People who bought this also bought…” Amazon-style recommendation

People

User_Id	name	Gender	Age
00	Neo	M	29
01	Trinity	F	28
02	Reagan	M	32
03	Agent Smith	M	30
04	The Architect	M	35
05	Morpheus	M	69

Friendship

ID	Friend since	relation	User1_Id	User2_Id
1	1990	knows	00	05
2	1991	Loves	01	00
3	1992	knows	02	01
4	1993	Coded by	03	02
5	1995	knows	05	06
6	1996	knows	05	04

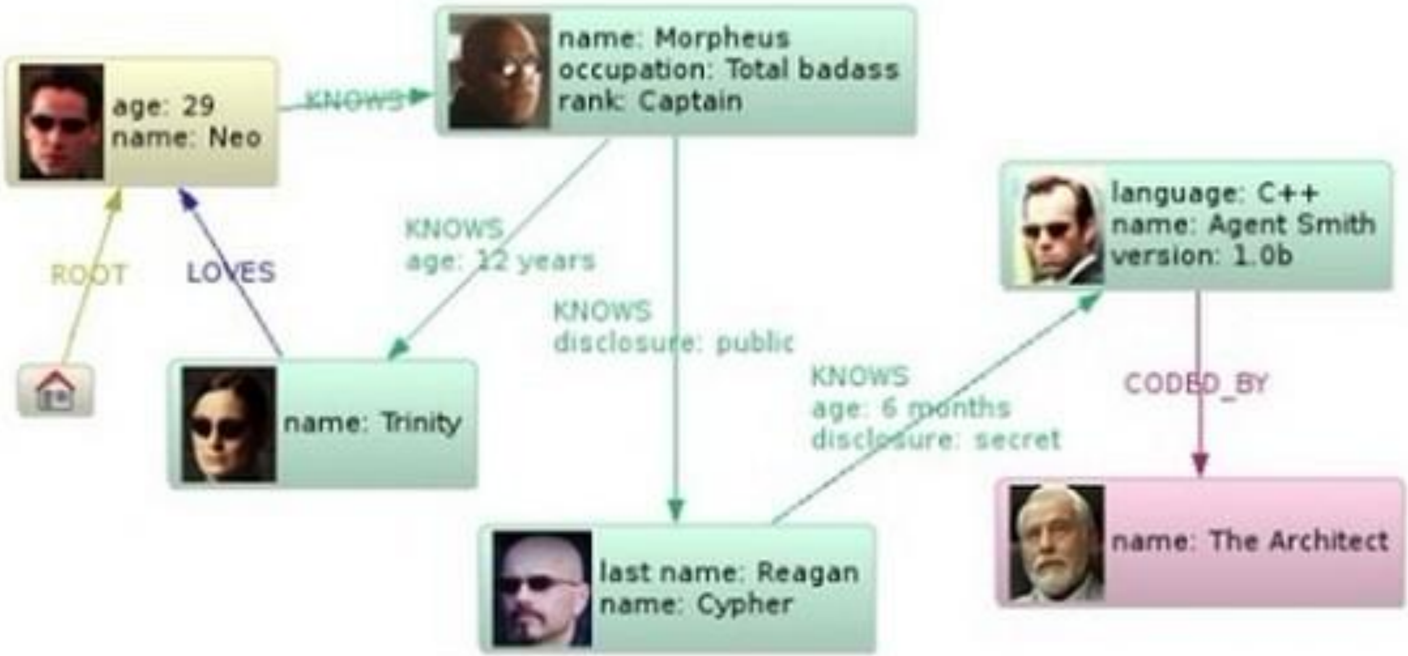


Image source: http://readwrite.com/2011/04/20/5-graph-databases-to-consider *dex

Players in the Field

OrientDB

- Pregel (Google)
- TAO (Facebook)
- FlockDB (Twitter)
- GraphLab (CMU)
- GraphChi (CMU)
- Neo4j (neotechnology)
- …

InfiniteGraph

Neo4j

The World's Leading Graph Database

Neo4j is an open-source, high-performance, enterprise-grade NOSQL graph database.

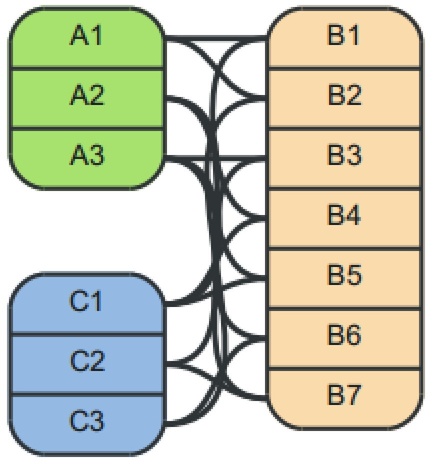
ACID
Java

http://www.neotechnology.com/customers/

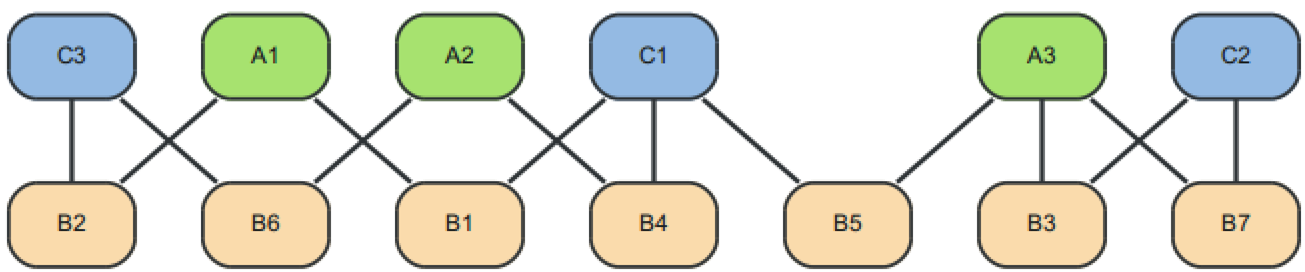
RDBMS and Graph Database

An Example:

RDBMS



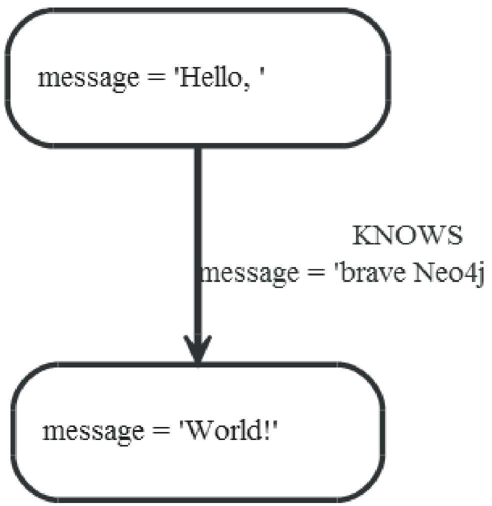
Graph Database



Neo4j code example

```
firstNode = graphDb.createNode();
firstNode.setProperty( "message", "Hello, " );
secondNode = graphDb.createNode();
secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
relationship.setProperty( "message", "brave Neo4j " );
```



Transaction, Index, …

```
nodeIndex = graphDb.index().forNodes( "nodes" );
Transaction tx = graphDb.beginTx();
try
{
    // Create users sub reference node
    Node usersReferenceNode = graphDb.createNode();
    graphDb.getReferenceNode().createRelationshipTo(
        usersReferenceNode, RelTypes.USERS_REFERENCE );
    // Create some users and index their names with the IndexService
    for ( int id = 0; id < 100; id++ )
    {
        Node userNode = createAndIndexUser( idToUserName( id ) );
        usersReferenceNode.createRelationshipTo( userNode,
            RelTypes.USER );
    }
```

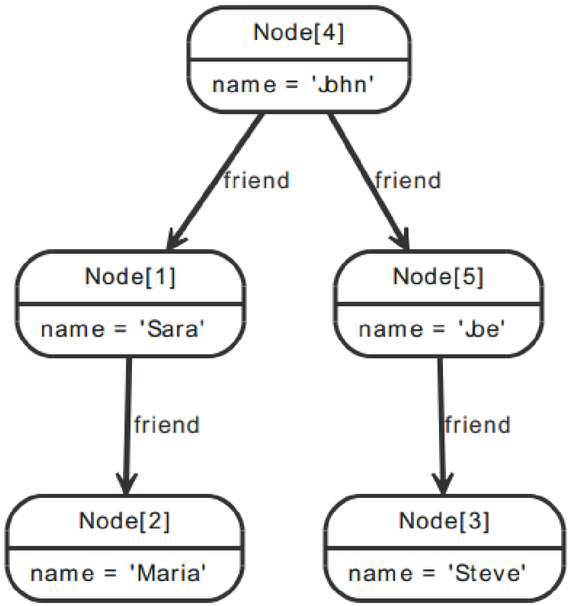
Create Index

Begin Transaction

```
int idToFind = 45;
Node foundUser = nodeIndex.get( USERNAME_KEY,
    idToUserName( idToFind ) ).getSingle();
System.out.println( "The username of user " + idToFind + " is "
    + foundUser.getProperty( USERNAME_KEY ) );
```

Index loopup

Cypher Query Language

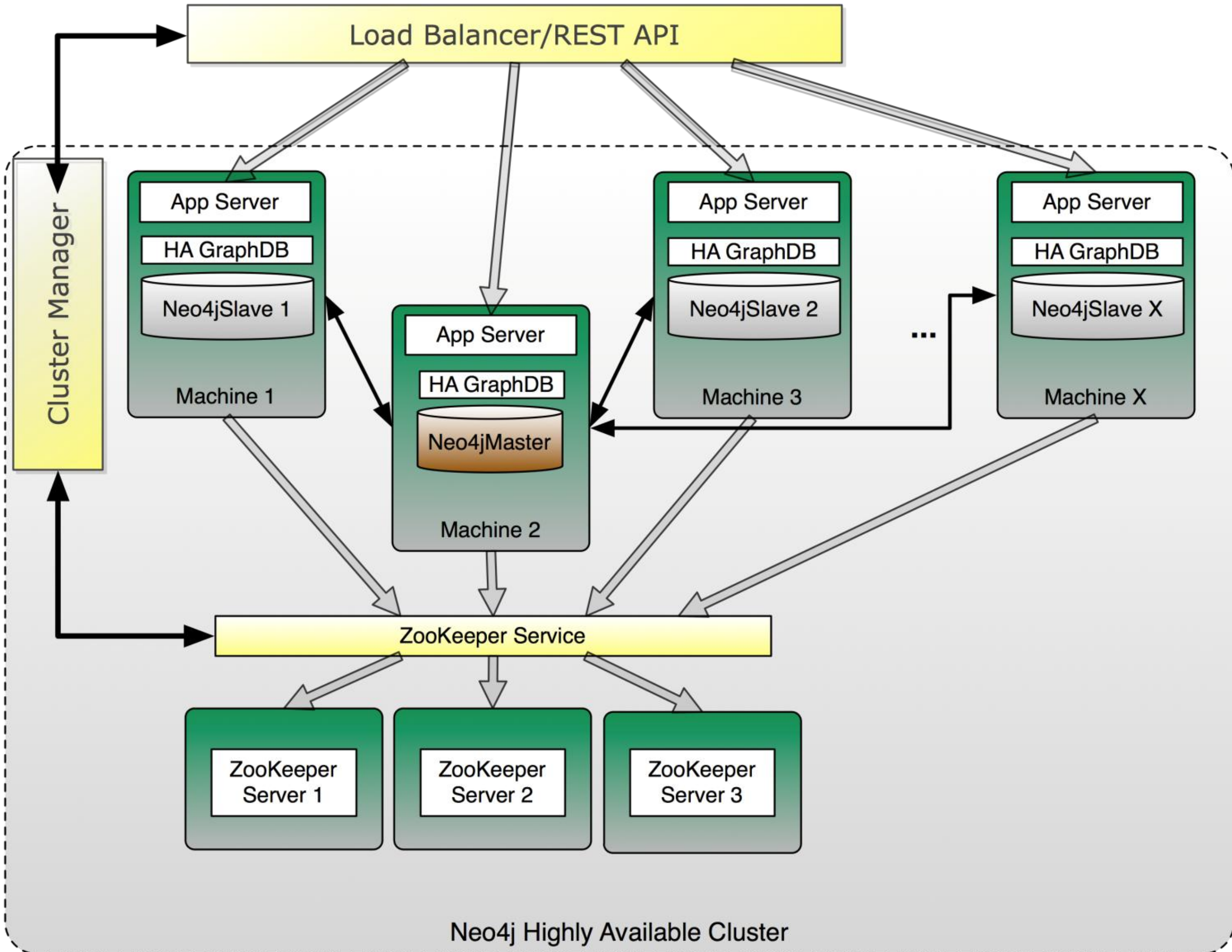



```
START user=node(5,4,1,2,3)
MATCH user-[:friend]->follower
WHERE follower.name =~ 'S.*'
RETURN user, follower.name
```

Resulting in:

user	follower.name
Node[5]{name:"Joe"}	"Steve"
Node[4]{name:"John"}	"Sara"
2 rows	
2 ms	

Neo4J Architecture



Neo4j High Availability

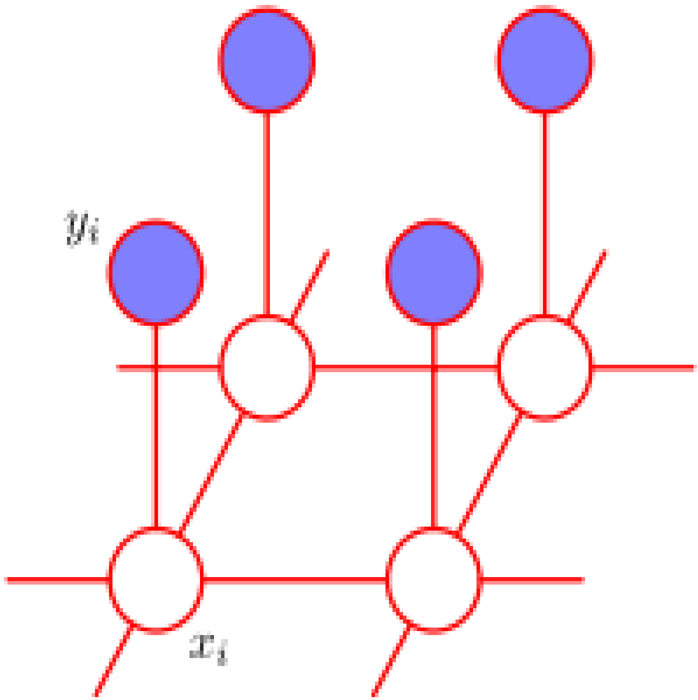
- Enterprise Edition only
- Read: Slaves are replicas of the master, therefore the whole system can handle more read operations than a single server - horizontal scaling
- Write: Any slave can handle write operation. Writing on slave will synchronize with the master (locking?); writing on master will synchronize to slaves – eventual consistency (configurable)

GraphLab

System for performing parallel (machine learning) graph algorithms

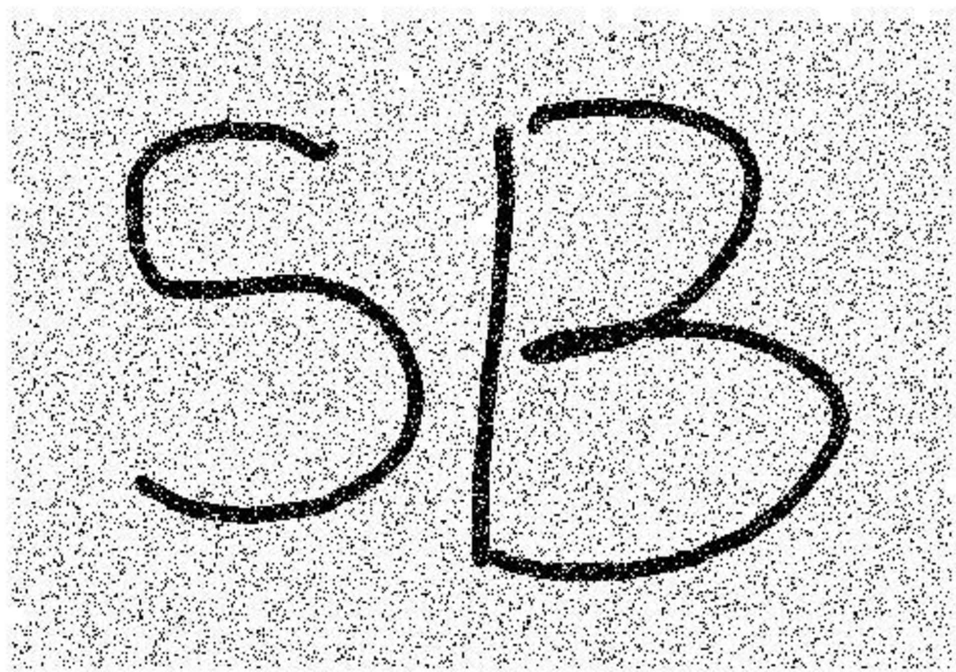
Multi-processor/cluster setting -- NOT fault tolerant or distributed (newer paper adds these features...)

Example: Pairwise MRF



source: http://grapeot.me/?tag=/markov+random+field

Example: Pairwise MRF



source: http://grapeot.me/?tag=/markov+random+field

Example: Loopy BP

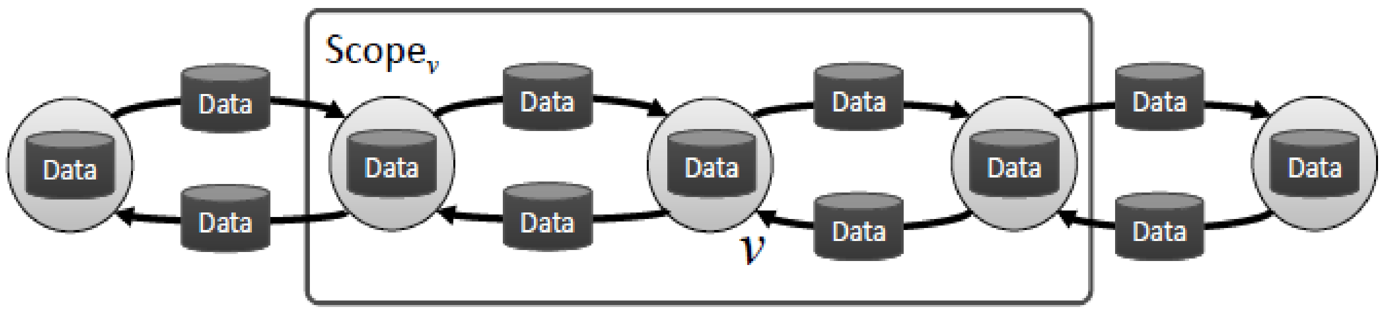
- Loopy Belief Propagation on Pairwise Markov Random Field.
- A message passing algorithm for performing inference on graphical models.
- Calculates the marginal distribution for each unobserved node, conditional on any observed nodes.

User Defined Computation

- Update Functions
  - Defines the local computation
- Sync Mechanism
  - Defines global aggregation

Update Function

- Operates on the data associated with small neighborhoods(scope) in the graph
- Scope: one vertex, its adjacent edges and neighboring vertices



(a) Scope

-

Yucheng Low et al 2010

Update Function

- Read-only access to the Shared Data Table.

Application of the update function to the vertex

$$D_{S_v} \leftarrow f(D_{S_v}, T)$$

- A GraphLab program may consist of multiple update functions
- Scheduling Model decides which updates functions apply to which vertices.

Sync Mechanism

- Aggregates data across all vertices in the graph
- The result is associated with a particular entry in the Shared Data Table.
- User provides a key k d an initial value r r_k

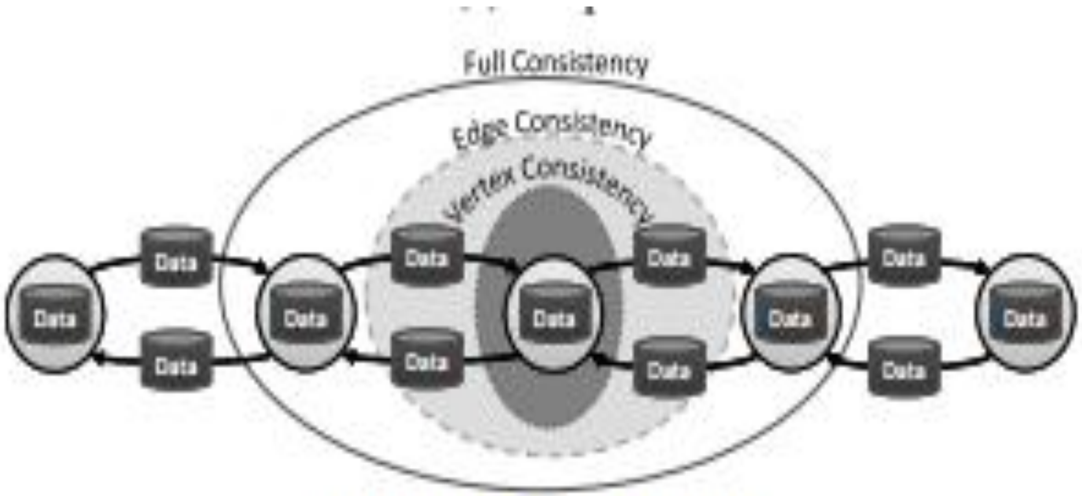
$$
\begin{aligned}
r_k^{(i+1)} &\leftarrow \mathrm{Fold}_k\left(D_v, r_k^{(i)}\right) \\
r_k^{l} &\leftarrow \mathrm{Merge}_k\left(r_k^{i}, r_k^{j}\right) \\
\mathbf{T}[k] &\leftarrow \mathrm{Apply}_k(r_k^{(|V|)})
\end{aligned}
$$

- Yucheng Low et al 2010

Data Consistency

- The simultaneous execution of two update functions can result in data inconsistency or corruption
- GraphLab provides a choice of three data consistency models, which enable user to balance performance and consistency

Data Consistency Models



(b) Consistency Models

Data Consistency Models

- Full consistency
  - Parallel execution may only occur on vertices that do not share a common neighbor
- Edge consistency
  - Parallel execution may only occur on non-adjacent vertice
- Vertex consistency
  - During the execution of f(v), no other function will be applied to v

Sequential Consistency

- A GraphLab program is ***sequentially consistent*** if for every parallel execution, there exists a sequential execution of update functions that produces an equivalent result

Sequential consistency guaranteed if:

- 1. The full consistency model is used
- 2. The edge consistency model is used and f(v) does not modify v's neighbors
- 3. The vertex consistency model is used and f(v) only accesses v's data

Scheduling

- ***Update schedule*** describes the order in which update functions are applied to vertices
- Represented by a parallel data structure called the ***scheduler***
- GraphLab provides several degrees of scheduling control

Base Schedulers

- Synchronous scheduler
  - All vertices updated simultaneously
- Round-robin scheduler
  - Vertices updated sequentially, using most recently available data

Task Schedulers

Permit update functions to add and/or reorder tasks

- FIFO schedulers
  - Permit task creation, not reordering
- Prioritized schedulers
  - Permit task creation and reordering

Set Scheduler

- User specifies a sequence of vertex set and update function pairs:

  $((S_1, f_1), (S_2, f_2), ..., (S_k, f_k))$

for i = 1...k do:

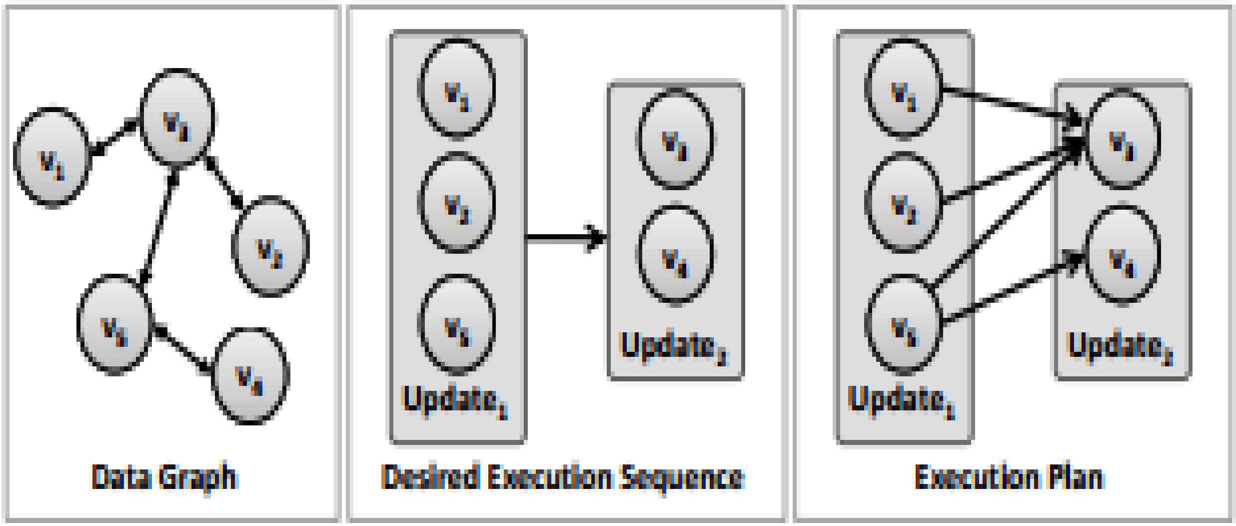
  Execute f_i on all vertices in S_i in parallel

  Wait for all updates to complete

Set Scheduler Execution Plan

- Waiting for all updates in i'th iteration to complete before moving onto (i+1)'th iteration = latency
- Execution plan improves this
  - Rewrite execution sequence as DAG
  - Vertex represents update task, edge represents execution dependency
  - Execute update tasks greedily

Set Scheduler Execution Plan



Termination Assessment

- Two methods:
  1. Scheduler: terminate when there are no remaining tasks
  2. SDT: terminate when shared data indicates convergence

GraphLab Summary

1. Data graph -- represents data and computational dependencies
2. Update function -- local computation
3. Sync mechanism -- aggregates global state
4. Consistency model -- determines how parallel
5. Scheduling primitives -- expresses order of computation
6. Termination conditions -- halts program

Summary

- Neo4j
  - Good for: ACID graph DB
  - Limit: HA mode won't increase the capacity
- GraphLab
  - Good for: Multi-processor/cluster parallel computing
  - Limit: No fault tolerance, no shared memory in distributed environment