如何管理Facebook规模的大图数据

— Billion规模大图数据管理系统关键技术

肖仰华

GDM@FUDAN (http://gdm.fudan.edu.cn)

复旦大学
Acknowledgement

• Thanks to my cooperators from MSRA including Dr Haixun Wang and Dr Bin Shao

• Many slides come from
  – SIGMOD 2012 Tutorial
  – KDD 2012 summer school
  – CCF第33期前沿技术学习班
  – Internet
Outline

• **What is a graph?**
• Why graph is so important?
• Why managing graph is so challenging?
• How to manage big graphs?
  – General principle
  – From system perspective
  – From algorithm perspective
What is a graph?

• A collections of entities and relationship between them
  – Entity
  – Relationships

• Euler
  – Seven Bridges of Königsberg
Models of graphs

- Weighted graphs
- Directed graphs
- Probabilistic graphs
- Evolving graphs
Notations

- Vertices/Nodes
- Edges/arcs
- Neighbors of a vertex
- Degree of a vertex
- Subgraph
- Shortest path

Example graph

![Example graph diagram](image)
Representation of a graph

- **Adjacent list**
  - Space efficient on sparse graph

- **Matrix**
Real graphs

- **Entity**:
  - Internet
  - Web
  - Power grid
  - Sensor network

- **Relationship**:
  - Protein interaction network
  - Gene regulatory network
  - Metabolic network

- **Social Network**:
  - Social relationships among individuals and organizations
  - Global trade network in the International Economic System
  - Online social network

- **Concept Network**:
  - Entity
  - Relationship
  - Linked Data
Synthetic graphs

• Synthetic networks
  – ER random network
  – Scale free network
  – Hierarchical network
Outline

• What is a graph?
• Why graph is so important?
• Why managing graph is so challenging?
• How to manage big graphs?
  — General principle
  — From system perspective
  — From algorithm perspective
Why graph is so important?

Graphs are ubiquitous

- Connectivity is a big trend
- Graph is a universal model

Semantic/information is connectivity
Graphs are ubiquitous

- Biological Network
- Social Network
- Transportation Network
- Program Flow
- Protein Structure
Social Graphs
Connectivity is a big trend

互联成为了21世纪人类社会的典型特征之一

电信、电话网络
电信、移动、网通...

基础设施网络
城市交通网络、供水网络、电网...

社会网络
科学家合作、Facebook，人人，开心...

物联网(Internet of things)
RFID，传感器网络

2012/11/21

Email: shawyh@fudan.edu.cn
Graph is a universal model
Outline

• What is a graph?
• Why graph is so important?
• *Why managing graph is so challenging?*
• How to manage big graphs?
  – General principle
  – From system perspective
  – From algorithm perspective
Goals

Build a general purpose graph system to manage and mine real big graphs!!
Challenges

• Challenges caused by real graphs
• Challenges of a general purpose graph system
Challenges caused by real graphs

Big graph

Complex graph operations

Complex graph

Cloud graph
Real Graphs are Big

Scaling up to big graphs is challenging!
Real Graphs are Complex

- Skewed degree distribution
- Unclear community structure
- Small world

Email: shawyh@fudan.edu.cn
Skewed degree distribution

Balanced load distribution is challenging!
Small world

- Six-degree separation
- Even smaller on social media network
- Graphs are becoming denser (kleinbeger 07)

Exploration in graphs is costly!
Unclear community structure

Ideal Community Structure

Community Structure of Real Graphs

Newman 2006

<table>
<thead>
<tr>
<th>Network</th>
<th>Size $n$</th>
<th>$\text{Modularity } Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{GN}$</td>
<td>$\text{CNM}$</td>
</tr>
<tr>
<td>Karate</td>
<td>34</td>
<td>0.401</td>
</tr>
<tr>
<td>Jazz musicians</td>
<td>198</td>
<td>0.405</td>
</tr>
<tr>
<td>Metabolic</td>
<td>453</td>
<td>0.403</td>
</tr>
<tr>
<td>E-mail</td>
<td>1,133</td>
<td>0.532</td>
</tr>
<tr>
<td>Key signing</td>
<td>10,680</td>
<td>0.816</td>
</tr>
<tr>
<td>Physicists</td>
<td>27,519</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Poor data parallelism: Partitioning a graph is hard!
Real Graphs have Complex Operations

- **Online query processing**
  - Shortest distance/path query
  - Subgraph matching query
  - SPARQL query
  - ...

- **Offline graph analytics**
  - Ranking: PageRank, SimRank
  - Clustering
  - Classification
  - Community detection
  - ...

- **Other graph operations**
  - Graph generation, visualization, interactive exploration, etc.

Graph computations are diversified!
Graph Operations need Random Access

• Data access along edges
  – Accessing a node’s neighbor requires “jumping” on graphs.
  – Consider DFS traverse
    • Hard to employ the locality of the graph

• Random access on disk or network is costly!
Order-dependent logic

• Process vertex in a dynamic order
  – Consider Dijkstra algorithm,
    • Always select the unvisited vertex that has the minimal weight to the visited nodes
  – Consider DFS
    • The next vertex to be visited is dependent on the previously visited node sequence

• Poor computation parallelism!
• Hard to be parallelized!
Graph processing without a system is hard!

**Fundamental issues**
- scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

**Architectural issues**
- Flynn’s taxonomy (SIMD, MIMD, etc.),
- network typology, bisection bandwidth
- UMA vs. NUMA, cache coherence

**Different programming constructs**
- mutexes, conditional variables, barriers, …
- masters/slaves, producers/consumers, work queues, …

**Different programming models**
- Message Passing
- Shared Memory

**Common problems**
- livelock, deadlock, data starvation, priority inversion…
- dining philosophers, sleeping barbers, cigarette smokers, …

**Programmer shoulders the burden of managing all these subtle issues...**
Benefits of a general purpose system

• Enable applications to focus on *algorithm* rather than system implementation

• Support *a variety of algorithms* on the same graph data in the same platform

• Support *various types of graphs*
Current Status of Graph Systems

• Good systems for processing *graphs*:
  – PBGL, Neo4j

• Good systems for processing *large data*:
  – Map/Reduce, Hadoop

• Good systems for processing specialized *large graph data*:
  – Specialized systems for pagerank, etc.
Current Status of Graph Algorithms

• Good solutions for small or intermediate size (million-nodes) graph:
  – We need to process billion-node graphs

• Good solution for serial algorithms
  – We need parallelized solutions

• Good solution for regular graphs
  – We need to handle irregular graphs
This is hard.

No good system for processing general large graphs
Taming big graph processing

• Our solution
  – Memory cloud: Trinity
  – Leverage properties of real graphs
    • Scale free
    • Small world
    • Community structure
  – Lightweight model and algorithm
    • Clustering instead of partitioning
    • Approximate distance oracle instead of accurate shortest path query
    • Repartitioning instead of direct partitioning
Outline

• What is a graph?
• Why graph is so important?
• Why managing graph is so challenging?
• How to manage big graphs?
  – General principle
  – From system perspective
  – From algorithm perspective
General Principle
System Design Choice

• Storage
  – Disk-based storage vs. Memory-based storage

• Scale-“UP” vs. Scale-“OUT”
  – Supercomputer vs. distributed cluster
Technical Trend

• Now: A commodity PC server has dozen gigabytes of memory

• Future: High-capacity memory and all-in-memory applications
### Trend in Cost of Memory Storage

<table>
<thead>
<tr>
<th></th>
<th>Today</th>
<th>In 5-10 years</th>
</tr>
</thead>
<tbody>
<tr>
<td># servers</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>GB/server</td>
<td>64GB</td>
<td>1024GB</td>
</tr>
<tr>
<td>Total capacity</td>
<td>64TB</td>
<td>1PB</td>
</tr>
<tr>
<td>Total server cost</td>
<td>$4M</td>
<td>$4M</td>
</tr>
<tr>
<td>$/GB</td>
<td>$60</td>
<td>$4</td>
</tr>
</tbody>
</table>

Adapted from: John Ousterhout, RAMCloud, 2010
Total Cost of Ownership

Query Rate ( Millions/sec )

Dataset Size (TB)

Hard disk

Flash Memory

RAM

Billion Scale Graph

Graphical representation showing the relationship between query rate and dataset size for different storage types: Hard disk, Flash Memory, and RAM.

Reproduced from Anderson’s SOSP 2009 paper.
Design Choice: Storage

1. Massive random data access pattern
2. lower total cost of ownership

*RAM can be ideal main storage for graphs*
Design Choice: Scale-up vs. Scale-out

• **Supercomputer model**
  – Programming model simple and efficient
    • shared memory address space
  – Expensive and not common
  – Hardware is your ultimate limit

• **Distributed cluster model**
  – Programming model is complex
    • Message passing and synchronization is more complex
  – Relatively cheaper and can make use of commodity pc
  – More flexible to meet various needs
Outline

• What is a graph?
• Why graph is so important?
• Why managing graph is so challenging?
• How to manage big graphs?
  — General principle
  — From system perspective
  — From algorithm perspective
Existing Systems

• Systems specialized for certain graph operations:
  – PageRank

• Systems designed for non-graph data
  – RDBMS
  – Map Reduce

• Graph Systems
  – Libraries for graph operations, e.g. PBGL
  – Matrix-based graph processing system, e.g. Pegasus
  – Graph database, e.g. Neo4j, HypergraphDB, Trinity
  – Graph analytics system, e.g. Pregel, Trinity, Giraph,
RDBMS

• Mainstay of business
  – Strong data integrity guarantee via ACID transactions

• Support complex queries
  – Standard query language: SQL

• Limited graph query support
  – Oracle (RDF query support)
Traverse Graph Using Joins

Get Neighbors of N1 that is 3-hops away

```
SELECT *
FROM N
LEFT JOIN E1 ON N.ID = E1.src
JOIN E2 ON E1.dst = E2.src
JOIN E3 ON E2.dst = E3.src
WHERE E.src = 1;
```

How about k-hops away?
Graph in the Jail of RDBMS

- The commonest graph operation “traversal” incurs excessive amount of table joins

- RDBMS: Mature but not for graphs
Map Reduce
Map Reduce

• High *latency*, yet high *throughput* general purpose data processing platform

• Optimized for *offline analytics* on large data partitioned on hundreds of machines
Map Reduce

- De facto of distributed large data processing

- Great scalability: supports extremely large data
Processing Graph using Map Reduce

• No online query support

• The map reduce data model is not a native graph model
  – Graph algorithms cannot be expressed intuitively

• Graph processing is inefficient on map reduce
  – Intermediate results of each iteration need to be materialized
  – Entire graph structure need to be sent over network iteration after iteration, this incurs huge unnecessary data movement
    • Huge Disk/Network IO, not suitable for random access on graphs
Pregel
Basic Idea: Think Like a Vertex!

Mapping from graph nodes to virtual BSP processors

Local Computation

Global Communication

Barrier Synchronization
Computation Model

• Graph computation is modeled as many supersteps

• Within each superstep
  – Each vertex reads messages sent in previous superstep
  – Each vertex performs computation in parallel
  – Each vertex can send messages to other vertices in the end of an iteration

• At the end of each superstep, Barrier synchronization
Example: SSSP in Pregel
Example: SSSP in Pregel

```
GDM@FUDAN http://gdm.fudan.edu.cn
Graph Data Management Lab at Fudan University

Example: SSSP in Pregel

8 -<-> -3-<-> 2
A   3 2 B
1 8
C
source
10
D
∞
```
Example: SSSP in Pregel

Graph:

- Source node labeled 0
- Node A connected to B with weight 2
- Node B connected to D with weight 10
- Node C connected to A with weight 8
- Node D connected to C with weight 1
- Node A connected to 5 with weight 3
- Node B connected to 2
- Node C connected to 9
- Node D connected to 12

Weights:

- Source to A: 1
- Source to B: 2
- Source to C: 1
- Source to D: 10
Example: SSSP in Pregel

![Graph with edge weights and source node labeled as 0]
Example: SSSP in Pregel

A

B

C

D

source

5

2

0

6

7

1

8

3

2

10

\infty

\infty

\infty

\infty
Pregel vs. Map Reduce

• Exploits fine-grained parallelism at node level

• Pregel doesn’t move graph partitions over network, only messages among nodes are passed at the end of each iteration

• Not many graph algorithms can be implemented using vertex-centric computation model elegantly
Pegasus: Peta-Scale Graph Mining
Pegasus

• An open source large graph mining system
  – Implemented on Hadoop

• Key Idea
  – Convert graph mining operations into iterative matrix-vector multiplication
Data Model of Pegasus

• Use vectors and matrices to represent graphs

• Specifically:
  – a graph with \( n \) vertices is represented by an \( n \times n \) matrix
  – each cell \((i, j)\) in the matrix represents an edge
    \((source = i, destination = j)\)
Example

Source: Pegasus, Kevin Andryc, 2011
Generalized Iterated Matrix-Vector Multiplication (GIM-V)

\[ M \times v = v' \quad \text{where} \quad v'_i = \sum_{j=1}^{n} m_{i,j} \times v_j \]

- Three primitive graph mining operations
  - \textit{combine2()}: Multiply \( m_{i,j} \) and \( v_j \)
  - \textit{combinAll}_i() : Sum \( n \) multiplication results
  - \textit{assign()}: Update \( v_j \)
Graph Mining in Pegasus

• Graph mining is carried out by customizing the three operations
  – combine2()
  – combineAll()
  – assign()
Example: Connected Components

Source: Pegasus, Kevin Andryc, 2011
Example: Connected Components

\[ c^{h+1} = M \times G c^h \]

Example: Connected Components

\[ c_i^{h+1} = \text{assign}(c_i^h, \text{combineAll}_i(\{x_j \mid j = 1..n, \text{ and } x_j = \text{combine2}_i(m_{i,j}, c_j^h)\))) \]
### Graph Mining

<table>
<thead>
<tr>
<th>Operations</th>
<th>Plain M-V</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diamet er</th>
<th>Conn. Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine2()</td>
<td>Multiply</td>
<td>Multiply</td>
<td>Multiply</td>
<td>Multiply bit-vector</td>
<td>Multiply</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with c</td>
<td>with c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>combineAll()</td>
<td>Sum</td>
<td>Sum with</td>
<td>Sum with</td>
<td>BIT-OR()</td>
<td>MIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prob. of</td>
<td>restart prob.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>random jump</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assign()</td>
<td>Assign</td>
<td>Assign</td>
<td>Assign</td>
<td>BIT-OR()</td>
<td>MIN</td>
</tr>
</tbody>
</table>

Source: Pegasus, Kevin Andryc, 2011
Pegasus

• Matrix based graph mining platform

• Support large scale graphs

• Many graph operations cannot be modeled by matrix-vector multiplications

• Matrix computation in general is costly
Trinity: Distributed Graph Engine
Trinity

- A distributed, in-memory key/value store
- A graph database for online query processing
- A parallel platform for offline graph analytics
All-in-memory General Purpose Graph Engine

- RAMCloud
- Pregel
- Trinity
- Graph Databases (e.g. Neo4j)
Graph Update: An extreme case

parallel execution in the entire cluster

graph type

data #:
10 B

type:
1 B
Case Study:
Trinity Graph Generation

Time (min)

0  2  4  6  8  10  12  14  16  18  20
1M  2M  4M  8M  16M  32M  64M  128M  256M  512M  1024M
Example: People Search Query

```
rosia david
He/She is your 2-hop friend.
facebook.com

david woods
He/She is your 2-hop friend.
facebook.com

lola david
He/She is your 2-hop friend.
facebook.com

david lapierre
He/She is your 2-hop friend.
facebook.com
```

```
david woods
He/She is your 2-hop friend.
facebook.com

bao david
He/She is your 2-hop friend.
facebook.com

david grado
He/She is your 2-hop friend.
facebook.com

david heredia
He/She is your 2-hop friend.
facebook.com
```
Cell Structure

- GID
- Cell ID
- Topology Data
- Cell Structure
- Additional Data
• Mapping
  – a field can reside on Trinity, on RDBMS, or on both

• Index
  – option of index: exact, full-text, spatial
Summary

• Building a general purpose large scale graph system is important but very challenging

• Memory-based scale-out approach is very promising
Outline

• What is a graph?
• Why graph is so important?
• Why managing graph is so challenging?
• How to manage big graphs?
  — General principle
  — From system perspective
  — From algorithm perspective
Our Focus

Key graph algorithms on various, large scale real graphs.
Outline

Graph partitioning
- On distributed machines

Graph Traversal
- BFS

Graph queries
- Shortest distance query

Graph analytics
- Betweenness computation

- Graph is big
  - billion node graphs

- Graph is distributed
  - Instead of centrically stored

- Solutions should be general enough to run on a general-purpose graph system
Graph partitioning
Graph Partitioning

• Problem definition
  – Divide a graph into $k$ almost equal-size parts, such that the number of edges among them is minimized.
  – Weighted variance

• Why?
  – Load balance
  – Reduce communication

• Example: BFS on the graph
  – Best partitioning needs 3 remote accesses
  – Worst partitioning needs 17

• NP-complete [Garey74]
Existing solutions

- Classical partitioning algorithm
  - Swapping selected node pairs: KL [kernighan 72] and FM [Fiduccia 82]
  - Simulated annealing based solutions [Johnson89]
  - Genetic algorithms [Bui96]
  - For small graphs (but with high quality)
- Multi-level partitioning solutions
  - METIS [Karypis95], Chaco [Hendrickson] and Scotch [Pellegrini96]
  - For million-node graphs
- Parallelized Multi-level partitioning solutions
  - ParMetis [Karypis98] and Pt-Scotch [Chevalier08]
  - For at most tens-of-million-node graphs.

- No good solutions for partitioning billion-node graphs on a general-purpose distributed system.
Graph partitioning in current systems

• Default: *Random partitioning*
  – PBGL, Pregel, neo4j, InfiniteGraph, HyperGraphDB
  – Quality is significantly worse than a refined method

• No partitioning algorithm is supported at the system level

• PBGL and Pregel support *user-defined partitioning*
KL

• Iteratively bisect a graph into two parts with equal size so that the number of cross edges is minimized

• In each bisection
  – Heuristically select the vertex pair whose swap leads to the most gain

• The gain of to swap two vertices a and b that lie in different parts,
  – \( g(a,b) = D_a + D_b - c_{a,b} \)
  – \( D_x = E_x - I_x \)
  – \( E_x \) is the number of neighbors of \( x \) on the different machine
  – \( I_x \) is the number of neighbors of \( x \) on the same machine
  – \( c_{a,b} \) is 1 iff a and b are neighbors.
State-of-art method: Metis

• A multi-level framework
• Three phrases
  – Coarsening by maximal match until the graph is small enough
  – Partition the coarsest graph by KL algorithm [kernighan 72]
  – Uncoarsening

Ref: [Metis 1995]
Maximal Matching

• A matching of a graph $G(N,E)$ is a subset $E_m$ of $E$ such that no two edges in $E_m$ share an endpoint.

• A maximal matching of a graph $G(N,E)$ is a matching $E_m$ to which no more edges can be added and remain a matching.

• Simple greedy algorithm
Maximal Matching - Example

How to coarsen a graph using a maximal matching

\[ G = ( N, E ) \]

\( E_m \) is shown in red

Edge weights shown in blue

Node weights are all one

\[ G_c = ( N_c , E_c ) \]

\( N_c \) is shown in red

Edge weights shown in blue

Node weights shown in black
Coarsening in Metis

- Assumption of coarsening: An optimal partitioning on a coarser graph is a good partitioning in the finer graph.

- It holds *only when node degree is bounded* (2D, 3D meshes).

- But real networks have *skewed degree distribution*.

- Metis uses refinement in the uncoarsening.
Possible solution: Multi-level Label Propagation

- Coarsening by label propagation
  - In each iteration, a vertex takes the *label that is prevalent in its neighborhood* as its own label.

- Lightweight
  - Easily implemented by message passing
  - Easily parallelizable

- Effective (for real networks)
  - Capable of discovering inherent community structures
How to handle imbalance?

• Imbalance caused by label propagation
  – Too many small clusters
  – Some extremely large clusters

• Possible solution:
  – First, limit the size of each cluster
  – Second, merge small clusters by
    • multiprocessor scheduling (MS)
    • weighted graph partitioning (WGP)

• For details:
  – How to partition a billion node graph, Microsoft Technical Report, 2012
New partitioning model on big graphs

• Repartitioning
  – Input: an existing partitioning and a graph
  – Output: a new partitioning with minimal edge cut that overlap significantly with the given partitioning

• Partitioning with replication
  – A balanced graph a partitioning but allowing overlapping, with constraint on the maximal number of replicated vertices
  – Side-effect: inconsistency

• Streaming graph partitioning (SGP)
  – Graph comes as streams

Ref: Feng 2012, Distributed Big Graph Storage, CCF Communications,
Greedy solution of SGP

• For every incoming vertex, select one of the machine that can maximize the gain function

• Let v be the incoming vertex, the gain function of machine m is:
  – \( F(m) = a(m)d(v) \)
  – Quality: \( d(v) \) is the number of v’s neighbors in m
  – Balance: \( a(m) \) is a punishing function of m’s used capacity p
    • Linear function: \( 1 - \frac{p}{c} \)
    • Exponential function: \( 1 - \exp(p-c) \)

  – Deterministic greedy selection with a linear \( a(m) \) practically produces the best partitioning (Stanton KDD 2012)
Distance Oracle
Why shortest distance query

• A basic graph operator
  – Used in many graph algorithms, including computing centrality, betweenness, etc.

• What is your Erdos number?
  – The shortest distance from you to mathematician Paul Erdos in the scientific collaboration network
  – Shortest distance can also be used to compute centrality, betweenness
Why distance oracle?

• Exact solutions
  – Online computation
    • Dijkstra-like algorithms, BFS
    • At least linear complexity, computational prohibitive on large graphs
  – Pre-compute all pairs of shortest path
    • Of quadratic space complexity

• Distance oracle
  – Report *approximate* distance between any two vertices in a graph in constant time by *pre-computation*
  – When graph is huge, approximation is acceptable, and pre-computation is necessary
Distance Oracle

• A pre-computed data structure that enables us to find the (approximate) shortest distance between any two vertices in constant time.

• Oracle construction
  – Space complexity
    • linear, or sub-linear
  – Time complexity.
    • Quadratic complexity is unaffordable

• Query answering
  – Time complexity
    • constant time.
  – Quality
    • Approaching exact shortest distances
Current Solutions

• Theoretic distance oracle with performance bound:
  – Thorup-and-Zwick’s distance oracle [Thorup01]
  – Reduce the construction time on weighted [Baswana06] or unweighted graphs [Baswana062],
  – Reduce space cost on power-law graphs [Chen09] or ER random graphs [Enachescu08].

• Practical distance oracle: heuristic approaches
  – Sketch based [Potamias09, Gubichev10, Sarma10, Goldberg05, Tretyakov11]
  – Coordinate based [zhao2010,zhao2011]
Thorup-and-Zwick’s distance oracle

- $O(kn^{1+1/k})$ space, can be constructed within $O(kmn^{1/k})$ time
- Distance query can be answered in $O(k)$ time with at most $2k - 1$ multiplicative distance estimation.

- When $k = 1$, the distance oracle returns the exact distance but occupies quadratic space
- When $k = 2$, the worst distance is 3-times of the exact distance and the oracle occupies $O(n^{1.5})$ space
Coordinate-based solution

- **Idea**
  - Mapping all vertices into a hyperspace
  - Use the distance of two vertices in the new space to approximate the shortest distance in the graph

- **Steps**
  - Choose several landmarks (~100)
    - **Heuristics:** Degree, betweenness, ...
  - Precisely calculate the distance from each landmark to all other vertices by **BFS starting from each landmark**
  - Calculate the coordinates of landmarks by **simplex downhill** according to the precise distance among landmarks
  - Calculate the coordinates of other vertexes by **simplex downhill** according to the distance from these vertex to each landmark

- **Advantage**
  - Constant query time
  - Linear space
  - Linear construction time

Reference: [zhao 2010, zhao2011]
How to learn the coordinate

- **Main method**: Simplex downhill
- **Objective function**
  - Minimize
    \[ \sum_{u,v \in \text{Landmark}} (\text{CoordinateDist}(c(u), c(v)) - \text{RealDist}(u, v))^2 \]
- **Space can be used**
  - Euclidean space [zhao 2010]
  - Hyperbololoid model [zhao 2011]

\[ \delta(x, y) = \text{arccosh} \left( \sqrt{\left(1 + \sum_{i=1}^{n} x_i^2\right)\left(1 + \sum_{i=1}^{n} y_i^2\right) - \sum_{i=1}^{n} x_i y_i} \right) \cdot |c| \]

- **Coordinate Dimension**: \( \sim 10 \)
Sketch-based Solution

• Basic idea
  – Create a sketch of bounded size for each vertex
  – Estimate the distance using the sketch

• Advantage
  – linear space
  – If the sketch encodes enough useful information, it can produce highly accurate answers in short time (in most cases in constant time).

• Procedure
  – select landmark set $L$
  – Generate the shortest distances from each landmark to any other vertex
    • Sketch[u] =\{(w_0, \delta_0), \ldots, (w_r, \delta_r)\}, where $w_i$ is a vertex (called seeds) and $\delta_i$ is the shortest distance between $u$ and $w_i$
  – Estimate the distance between vertices $u$ and $v$ by the minimal value of $d(u,w) + d(w,v)$ over all $w \in L$
Improvement on sketch-based solution

• Improve distance estimation
  – cycle elimination and tree-structured sketch [Gubichev10]
  – uses the distance to the least common ancestor of $u$ and $v$ in a shortest path tree [Tretyakov11]

• Improve landmark selection
  – optimal landmark selection is NP-hard
  – betweenness is a good landmark [Potamias09]
  – randomized seed selection [Sarma10]
Distance oracle summary

- Accuracy may be compromised
  - Node degrees, instead of their centrality, is usually used as a criterion for landmark selection to reduce the cost.

- Some distance oracles take very long time to create.
  - For example, in [Tretyakov11], it takes 23h to approximate betweenness for a 0.5 billion node graphs [Tretyakov11] even on a 32-core server with 256G memory.

- None of the previous distance oracles is designed for distributed graphs.
- Can hardly scale to billion node graphs on a general purpose graph system
Scale to billion node graphs with high accuracy

- Smart landmark selection  
  — in local graphs

- Smart distributed BFS

- Smart answer generation rule

For details:
Towards a Distance Oracle for Billion Node Graphs, Microsoft Technical Report, 2012
Distributed BFS
Asynchronized BFS

- Asynchronized BFS (Bellman-ford)
  - In each iteration, update each vertex’s distance as the minimal distance of its neighbors plus one
  - Any time when a vertex distance is updated, it will trigger all its neighbors to update their distance
  - Until no distance update

- $O(|V||E|)$, but flexible, allows fine-grained parallelism
Level-synchronized BFS [Andy05]

- Explore from a node level by level

- **Iterate:**
  - Each vertex of level $x$ sends distance update messages to their neighbors
  - Each vertex waits for messages for itself. If its distance is still unknown, update its distance as $x+1$
  - *Synchronize at the end of each level*

- O(E) complexity
Possible Optimizations

• Observation:
  – Vertices of large degree are frequently visited even their distances to the source have already been computed.

• Optimization:
  – Cache the distances of large degree nodes on each machine

• “80/20” rule in real graphs

• Results on Trinity
  – 50% savings

• Scale-free graph

\[ d(v) = \frac{1}{N^R} r(v)^R \]

For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012
Betweenness computation
Betweenness computation

- Betweenness counts the fraction of shortest paths passing through a vertex

- Applications of betweenness
  - Vertex importance ranking
  - Landmark selection in distance oracle
  - Community detection

\[ C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(u)}{\sigma_{st}} \]

- Where, \( \sigma_{st} \) denotes the number of shortest paths from \( s \) to \( t \), \( \sigma_{st}(u) \) denote the number of shortest paths from \( s \) to \( t \) that pass through \( u \)
Betweenness computation on large graphs

- Exact betweenness costs $O(nm)$ time, unacceptable for large graphs [Brandes01]

- Lightweight approximate betweenness
  - Count shortest paths rooted at sampled vertices [Ercsey10]
  - Count shortest path with limited depth [Bader06][Brandes07]

- Parallelized exact betweenness [Bader06, Madduri09, Edmonds10, Tan09, Tu09]
  - On massive multithreaded computing platform [Bader06, Madduri09],
  - On distributed memory system [Edmonds10]
  - On multi-core systems [Tan09, Tu09].
Local Graph, Extended Local Graph, and Graph
A lightweight Distributed betweenness approximation

- On distributed platform
  - Compute the shortest paths within each machine
  - Local shortest path with two ends both in the local graph has high quality
  - Sample the local shortest with the probability proportional to its quality
  - Use the high-quality local shortest path to approximate exact betweenness

For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012
Summary of key algorithms

- Most existing graph solutions are designed for centralized platforms
- Power of distributed computing has not been fully exploited for solving challenging graph problems
- Properties of distributed graphs have not been fully explored yet
- Previous existing solutions can hardly scale to real large graphs, especially billion node graphs
References-1

References-2

- [andy05] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L, SC’05
- [Pellegrini96] F. Pellegrini et al., “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs,” in HPCN Europe ’96
- [Pregel] G. Malewicz et al., “Pregel: a system for large-scale graph processing,” in SIGMOD ’10
References-3

- [Bader06] D. A. Bader and K. Madduri, “Parallel algorithms for evaluating centrality indices in real-world networks,” in *ICPP ’06*.
- [Tu09] D. Tu and G. Tan, “Characterizing betweenness centrality algorithm on multi-core architectures,” in *ISPA’09*.
Thanks!
Who is GDM@FUDAN

- T-Robot
- Chinese Knowledge Graph
- Big graph management system
Framework

Web Resource

Entity/concept extraction  Entity resolution
Entity Evaluation  Relation Extraction

Knowledge Extraction

Applications

Reading  Search  QA

Conceptualization  Sense Disambiguation

Knowledge base operations

Cloud based Graph DB engine

Chinese Knowledge Graph
中文知识图谱

• “云时代的四库全书”
• “Root of china”中国之根
• 链接：
  http://gdm.fudan.edu.cn/Soso/index.jsp
Thanks for your attentions!!