Scalable Computing Frameworks for Graphs

Shangpu Jiang
Outline

- Introduction
- Systems
  - Pregel
  - GraphLab
- Applications
Introduction
Motivation

- Computations with large scale graphs are pervasive
  - Web (information retrieval, information extraction, …)
  - Social networks (link prediction, collaborative filtering, …)
  - Science (protein structure, …)
  - …
Possible solutions

- Develop distributed architecture per task
  - too much effort
- Existing general purpose platform (e.g., MapReduce)
  - works poorly
- Single-computer (parallel) graph algorithm libraries
  - limited scale
MapReduce: word count
MapReduce workflow
Why MapReduce fails

- Complex dependencies of data
- Iterative convergent process
- Intermediate results fit in memory
Map Reduce

Data-Parallel

Feature Extraction
Computing Sufficient Statistics

Cross Validation

Graph-Parallel

Lasso
Label Propagation
Belief Propagation

Kernel Methods

Tensor Factorization

Deep Belief Networks

PageRank

Neural Networks
Overview

❖ A google project Malewicz et al. [SIGMOD’10]
❖ Named after a river in Königsberg
❖ Apache Giraph, an open source project based on Pregel
Bulk Synchronous Parallel model
Distributed systems for graphs

http://stochastix.files.wordpress.com/
How do we program graph computations?

“Think like a vertex.”

—Malewicz et al. [SIGMOD’10]
BSP model for graphs

- Synchronous iterations (supersteps)
- Master initiates each superstep
- At every superstep S
  - workers asynchronously execute a user function on all vertices
  - vertices receive messages sent from inbound edges at superstep S-1
  - vertices modify their values
  - vertices send messages to outbound edges that will be read at superstep S+1
- vertices vote to halt
Example: find max

Superstep 0

\[
\begin{align*}
& \text{i}_\text{val} := \text{val} \\
& \text{for each message } m \\
& \quad \text{if } m > \text{val} \text{ then } \text{val} := m \\
& \quad \text{if } \text{i}_\text{val} == \text{val} \text{ then } \\
& \quad \quad \text{vote_to_halt} \\
& \text{else} \\
& \quad \text{for each neighbor } v \\
& \quad \quad \text{send_message}(v, \text{val})
\end{align*}
\]

Message from super-step (i-1)

Message to super-step (i+1)

Superstep 1

Superstep 2

Superstep 3
Local reduce: combiners
Global reduce: aggregators
Fault tolerance

- Check pointing
  - frequency determined by mean time to failure model
- Confined check pointing, only recover lost partitions
template<typename VertexValue,
    typename EdgeValue,
    typename MessageValue>

class Vertex {
    public:
        virtual void Compute(MessageIterator* msgs) = 0;

        const string& vertex_id() const;
        int64 superstep() const;

        const VertexValue& GetValue();
        VertexValue* MutableValue();
        OutEdgeIterator GetOutEdgeIterator();

        void SendMessageTo(const string& dest_vertex,
                           const MessageValue& message);

        void VoteToHalt();
};
Example 1: page rank
class PageRankVertex
    : public Vertex<double, void, double> {
public:
    virtual void Compute(MessageIterator* msgs) {
        if (superstep() >= 1) {
            double sum = 0;
            for (; !msgs->Done(); msgs->Next())
                sum += msgs->Value();
            *MutableValue() =
                0.15 / NumVertices() + 0.85 * sum;
        }
        if (superstep() < 30) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        } else {
            VoteToHalt();
        }
    }
};

\[ PR(p_i; t + 1) = \frac{1 - d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)} \]
Experiments

Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multicore machines

Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines
GraphLab
Overview

- **GraphLab** [Low et al. UAI’10]
- Distributed GraphLab [Low et al. VLDB’12]
- **PowerGraph** (for power-law graphs) [Gonzalez et al. OSDI’12]
- **GraphChi** (for a single machine) [Kyrola et al. OSDI’12]
Gather-apply-scatter abstraction

```
interface GASVertexProgram(u) {
    // Run on gather_nbrs(u)
    gather(D_u, D_{u,v}, D_v) → Accum
    sum(Accum left, Accum right) → Accum
    apply(D_u, Accum) → D_u^{new}
    // Run on scatter_nbrs(u)
    scatter(D_u^{new}, D_{u,v}, D_v) → (D_{u,v}^{new}, Accum)
}
```

**Algorithm 1: Vertex-Program Execution Semantics**

```
Input: Center vertex u

if cached accumulator a_u is empty then
    foreach neighbor v in gather_nbrs(u) do
        a_u ← sum(a_u, gather(D_u, D_{u,v}, D_v))
    end
end

D_u ← apply(D_u, a_u)

foreach neighbor v scatter_nbrs(u) do
    (D_{u,v}, Δa) ← scatter(D_u, D_{u,v}, D_v)
    if a_v and Δa are not Empty then
        a_v ← sum(a_v, Δa)
    else
        a_v ← Empty
    end
```
Asynchronous computation

- Pregel: BSP model
  - vertex computations are done simultaneously
- GraphLab: Asynchronous model
  - vertex computations are done sequentially
  - or exists a sequentially equivalence
Greedy graph coloring

```c
// gather_nbrs: ALL_NBRS
gather(D_u, D_{u,v}, D_v):
    return set(D_v)
sum(a, b): return union(a, b)
apply(D_u, S):
    D_u = min c where c ∉ S
// scatter_nbrs: ALL_NBRS
scatter(D_u, D_{u,v}, D_v):
    // Nbr changed since gather
    if(D_u == D_v)
        Activate(v)
    // Invalidate cached accum
    return NULL
```
Belief propagation

- Asynchronous belief propagation
- more difficult to converge on color boundaries
- dynamic scheduling
Synchronous vs Asynchronous

- **Bulk Synchronous Parallel model**
  - Jacobi iteration
  - Computation in phases
  - Simple to build
    - no worries about race conditions
    - simple to make fault-tolerant
  - Slower convergence
  - **Straggler problem**

- **Asynchronous model**
  - Gauss-Seidel iteration
  - Vertices see latest information from their neighbors
  - Hard to build
    - race conditions happen all the time
    - hard to make fault-tolerant
  - may need **scheduler**
  - Faster convergence
Scheduler

- FIFO scheduler
- Priority scheduler (e.g., residual BP)
- other schedulers (e.g., splash BP)
Messaging vs shared memory

- **Message passing**
  - simple to implement, no need for consistency model or concurrency control
  - **Passively** receives values from neighbors
  - higher overhead (storage, communication)

- **Distributed shared memory**
  - hard to implement, need to consider data consistency
  - **Actively** reads values, more flexible
  - more efficient
Data consistency

- Serializability: For each parallel execution, there exists a sequential execution that produces an equivalent result.
Data consistency

- Distributed locking engine
  - fully asynchronous
- Chromatic engine
  - partially asynchronous
- color-step
PowerGraph: Power law graphs

(a) Twitter In-Degree

\(\alpha = 1.7\)

(b) Twitter Out-Degree

\(\alpha = 2\)
Edge cut vs vertex cut

(a) Edge-Cut

(b) Vertex-Cut
Communication pattern

1. Gather
2. Accumulator (Partial Sum)
3. Apply
4. Updated Vertex Data
5. Scatter

Machine 1

Mirror

Gather
Scatter

Machine 2

Gather
Scatter
Comparison on power law graphs
Other systems

- Standford Graph Processing System (GPS)
- Apache Spark + GraphX
Applications
Graph analytics

- Shortest path
- Triangle counting
- Graph coloring (async)
Markov Random Field

- Belief propagation (sync, async: residual BP, splash BP)
- Dual decomposition
- Gibbs sampling (async)
Image stitching
Video co-segmentation

- Predict labels (sky, building, grass, pavement, trees)
- 32 frames, 120 x 50 superpixels, 192k vertices
- MRF + Gaussian Mixture Model
Collaborative filtering
Collaborative filtering

- Matrix (tensor) factorization
- Alternating least square (ALS), Stochastic gradient descent (SGD), …
- A bipartite graph of *users* and *movies* as vertices (feature vectors as vertex data), *ratings* as edges
Named Entity Recognition

- Co-EM

- A bipartite graph of **noun phases** and **contexts** as vertices, **co-occurrences** as edges.
Topic modeling

- Label topics for documents
- Latent Dirichlet Allocation
- Parallel collapsed Gibbs sampler
- A bipartite graph of documents and terms as vertices, occurrences as edges
Summary

- Distributed frameworks for graphs (Pregel, GraphLab, etc.)
- BSP vs. Asynchronous, message passing vs. distributed memory sharing, edge cut vs. vertex cut, …
- Applications esp. in Big Learning