GraphLab Parallel and Distributed Machine Learning

Yang Chen

yang@cise.ufl.edu

Computer and Information Science and Engineering
University of Florida

Jan 24, 2013

Outline

MapReduce: Execution Model

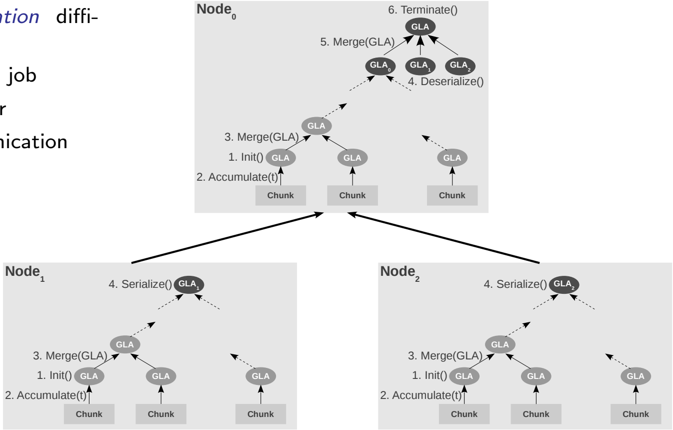
Problems with MapReduce

- Iterative algorithms → **Spark** [ZCF+10], **HaLoop** [BHBE10]
- Aggregations → **GLADE (Datapath)** [RD12, ADJ+10]
- Interactive data processing → **Spark**, **(Google) Dremel** [MGL+10]
- Graph processing → **(Google) Pregel** [MAB+10], **GraphLab** [LGK+10, LBG+12]

GLA vs MapReduce

Aggregate computation difficult to express:

- Extra merging job
- Singel Reducer
- Extra communication

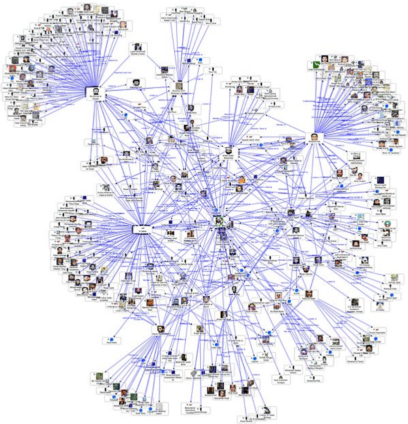


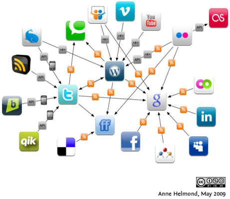
Datapath is an open-source project developed at UF[1].

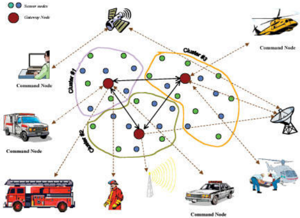
[1] http://datapath.googlecode.com

DSR @ UF Data Science Research

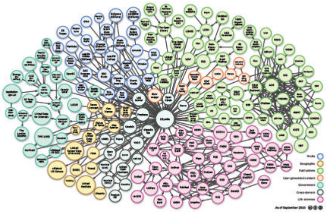
Graph Computations







Graph Computations



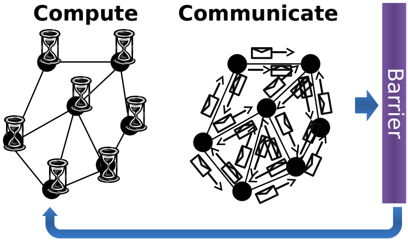
Why not MapReduce?

- Not good at graph algorithms (multiple stages $\rightarrow$ lots of overhead)
- Sometimes have to be bend and twist problems into unnatural forms.

Existing graph libraries? (e.g. LEDA, Boost graph library)

- Not scalable (billions of vertices)

Pregel



Pregel: Large-Scale graph processing at Google [MAB+10][2]:

- Bulk Synchronous Parallel model [Val90].
- Slow jobs slow down the whole process.

[2] For an open-source implementation, see Apache Giraph.

GraphLab

- GraphLab [LGK+10, LBG+12] is a parallel framework designed specifically for ML:
  - Graph dependencies
  - Iterative
  - Asynchronous
  - Dynamic
- Simplifies design of parallel programs:
  - Abstract away hardware issues
  - Automatic data synchronization
  - Addresses multiple hardware architectures

GraphLab

Data-Parallel ⟷ Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

GraphLab

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

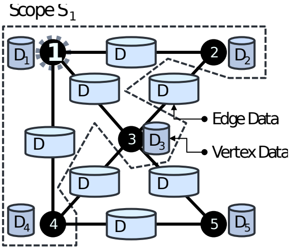
Data-Mining
PageRank
Triangle Counting

Outline

Data Graph

A data graph $G = (V, E, D)$ encodes the problem specific *sparse computational structure* and directly modifiable program state.

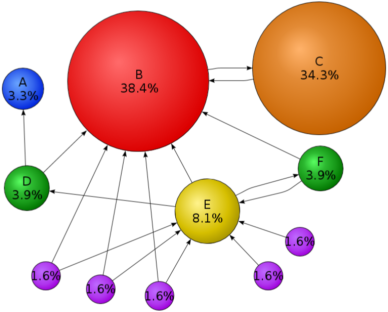
- Vertex data: $\{D_v | v \in V\}$
- Edge data: $\{D_{u \to v} | \{u, v\} \in E\}$
- v's scope $\mathcal{S}_v$ is defined to be v, its adjacent edges and neighboring vertices.



DSR @ UF Data Science Research

Example (PageRank)

- Each vertex corresponds to a web page.
- D_v stores R_v, the current estimate of the PageRank.
- $D_{u \to v}$ stores $w_{u,v}$, the directed weight of the link.



Update Functions

Update: $f(v, \mathcal{S}_v) \to (\mathcal{S}_v, \mathcal{T})$

- S_v: The scope of v, the data stored in v and its neighboring vertices and edges.
- $\mathcal{T}$: The task set, which includes future task executions. $\mathcal{T}$ contains tuples of the form (f, v).

The update function mechanism allows for asynchronous computation on the sparse dependencies defined by the data graph.

DSR @ UF
Data Science Research

Update Functions

Example (PageRank)

Algorithm 1 PageRank update function

Input: Vertex data $R(v)$ from $\mathcal{S}_v$
Input: Edge data $\{w_{u,v} | u \in \mathcal{N}(v)\}$ from $\mathcal{S}_v$
Input: Neighbor vertex data $\{R(u) | u \in \mathcal{N}(v)\}$ from $\mathcal{S}_v$
$R_{\text{old}}(v) \leftarrow R(v)$ ▷ Save old PageRank
$R(v) \leftarrow \alpha / n$
foreach $u \in \mathcal{N}(v)$ **do** ▷ Loop over neighbors
    $R(v) \leftarrow R(v) + (1 - \alpha \times w_{u,v} \times R(u)$
if $|R(v) - R_{\text{old}}(v) > \epsilon|$ **then**
    return $\{u | u \in \mathcal{N}(v)\}$ ▷ Schedule neighbors to be updated
Output: Modified scope $\mathcal{S}_v$ with new $R(v)$

Execution Model

Simple loop semantics

Algorithm 2 GraphLab Execution Model

Input: Data graph $G = (V, E, D)$
Input: Initial vertex set $\mathcal{T} = \{v_1, v_2, \ldots\}$
while $\mathcal{T}$ is not empty **do**
$\quad v \leftarrow \texttt{RemoveNext}(\mathcal{T})$
$\quad (\mathcal{T}', \mathcal{S}_v) \leftarrow f(v, \mathcal{S}_v)$
$\quad \mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
Output: Modified data graph $G = (V, E, D')$

Sequential Consistency Models

Definition (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.

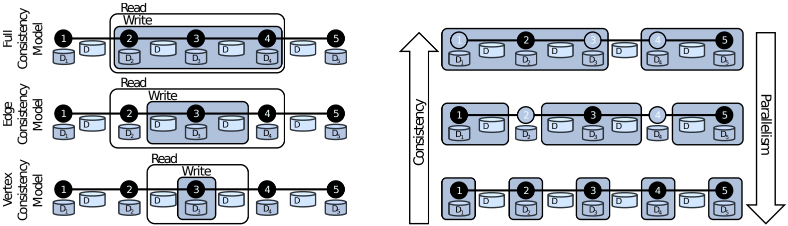
Proposition

GraphLab guarantees sequential consistency under the following three conditions:

1. The *full consistency* model is used.
2. The *edge consistency* model is used and update functions do not modify data in adjacent vertices.
3. The *vertex consistency* model is used and update functions only access local vertex data.

DSR @ UF
Data Science Research

Sequential Consistency Models



(a) Consistency Models

(b) Consistency and Parallelism

Figure: GraphLab consistency models and parallelism.

Sync Operation and Global Values

Definition (Sync operation)

The sync operation is defined as an associative commutative sum:

$$Z = \texttt{Finalize}\left(\bigoplus_{v \in V} \texttt{Map}(\mathcal{S}_v)\right)$$

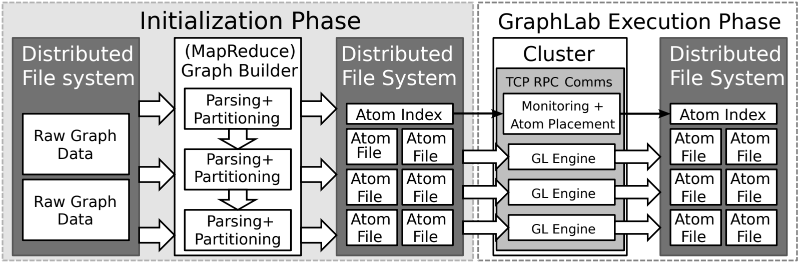
- Runs continuously in the background to maintain updated estimates of the global value.

Example

```
VoteToHalt();
```

Outline

Overview

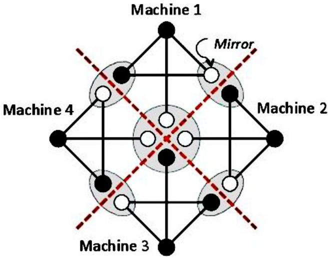


- Initialization: partitioning, loading
- Execution: scheduling, locking, fault tolerance

Distributed Data Graph

Two-phase partitioning

1. Over-partition into *atoms*.
2. Partition meta-graphs over physical machines



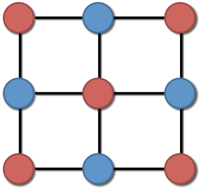
- The graph is partitioned over machines using *vertex separators*.
- Each vertex which span multiple machines, has a *master machine* (a black vertex), and all other instances of the vertex are *ghosts/mirrors*.
- The ghosts are used as *caches* for their true counterparts across the network.

DSR @ UF
Data Science Research

Distributed GraphLab Engine

- Chromatic engine
  - Each color-step executes vertices in same color.
  - Partially asynchronous.
- Locking engine
  - Non-blocking reader-writer lock to achieve pipelined locking.
  - Fully asynchronous.

The choice of execution engine affects performance and expressiveness.



Fault Tolerance: Distributed Checkpointing

Algorithm 3 Chandy-Lamport Distributed Snap-Shot

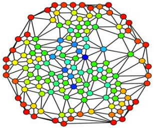
Input: vertex v
if v was already snapshotted **then**
    `Quit`
Save D_v ▷ Save current vertex
foreach $u \in \mathcal{N}(v)$ **do**
    if u was not snapshotted **then**
        Save D_{uv}
        `Schedule` u for a Snapshot Update
Mark v as snapshotted

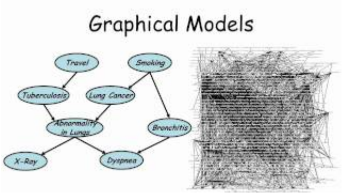
- Expressed as an update function.
- Guarantees a consistent snapshot under certain conditions.

Outline

Applications

Graph processing, collaborative filtering, graphical models, cloud computer vision, topic modeling, etc.[3]





[3] See http://graphlab.org for source code access.

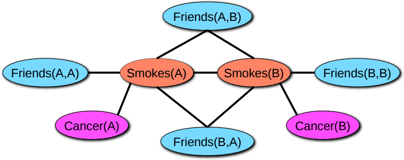
Knowledge Representation: Markov Logic Networks

We use *Markov logic networks* [RD06] to represent domain knowledge:

Weight	First-order logic
1.5	$\forall x \ \mathtt{Sm}(x) \rightarrow \mathtt{Ca}(x)$
1.1	$\forall x, y \ \mathtt{Fr}(x,y) \rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$

A set of constants $C = \{$**Anna** (A), **Bob** $(B)\}$

The Markov logic network is used to generate a *Markov network* (factor graph) [KF09, KFL01, WMM10] that defines a probability distribution among explicit and inferred knowledge.



DSR @ UF
Data Science Research

Markov Network Inferece

Marginal Inference–Computing Probabilities

- The Markov random field defines a probability distribution on all nodes in it:

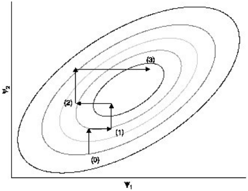
$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(\mathbf{x})\right),$$

  where n_i is the number of ground clauses that satisfy clause i in the MLN and w_i is the weight of that clause. Z is a normalizer, also called the *partition function*.
- In general, this is intractable due to Z. We use *sampling methods* to approximate the probabilities.

Gibbs Sampling–A Quick Primer

Suppose a random vector $(W, X, Y, Z) \sim f(w, x, y, z)$. The Gibbs sampler samples each of W, X, Y, Z in turn:

$$
\begin{aligned}
w_i &\sim f(w|x = x_{i-1}, y = y_{i-1}, z = z_{i-1}) \\
x_i &\sim f(x|w = w_i, y = y_{i-1}, z = z_{i-1}) \\
y_i &\sim f(y|w = w_i, x = x_i, z = z_{i-1}) \\
z_i &\sim f(z|w = w_i, x = x_i, y = y_i)
\end{aligned}
$$



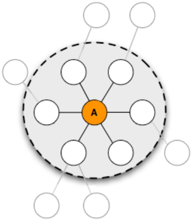
for i from 1 to a user-specified number N.

Gibbs Vertex Update Model

Proposition (Markov networks *locality property*)

A variable X_i is conditionally independent of all other variables given its neighbors (*Markov blanket*):

$$\pi(X_i|\mathbf{X}_{\mathcal{N}_i}) = \pi(X_i|\mathbf{X}_{-i})$$



Therefore, in each sample step, a vertex has to read from its neighbors and write to itself. This fits into GraphLab's *edge consistency model*.

The Chromatic Sampler

Algorithm 4 The Chromatic Sampler

Input: k-colored Markov network

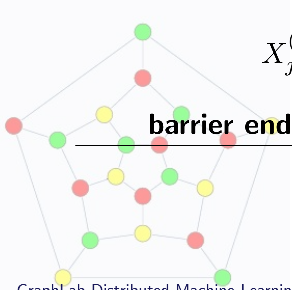
for Colors κ_i: $i \in \{1 \ldots k\}$ **do**

  for Variables $X_j \in \kappa_i$ in the i^{th} color **do in parallel**

    Ececute Gibbs Update:

$$X_j^{(t+1)} \sim f\left(x_j^{(t+1)} | \mathbf{x}^{(t+1)}_{\mathcal{N}_j \in \kappa < i}, \mathbf{x}^{(t)}_{\mathcal{N}_j \in \kappa > i}\right)$$

  barrier end



The Chromatic Sampler

Theorem

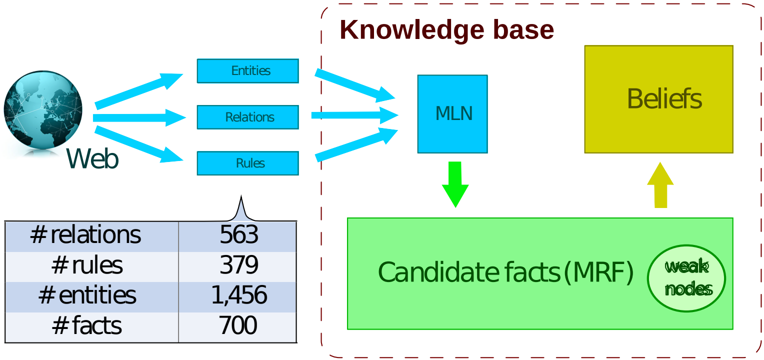
Given p processors and a k-coloring of an n-variable MRF, the Chromatic sampler is ergodic and generates a new joint sample in running time:

$$O\left(\frac{n}{p} + k\right)$$

Proof.

See [GLG+11]. $\square$

Experiments



# relations	563
# rules	379
# entities	1,456
# facts	700

# samples	10	100	200	500	State-of-Art
Sherlock-700 colors: 16	1.2s	12.6s	28.3s	65.1s	55min

[4] TUFFY, FELIX [NRDS11, NZRS11]

Natural Language Processing

A similar idea can be applied to *conditional random fields* [SM06].

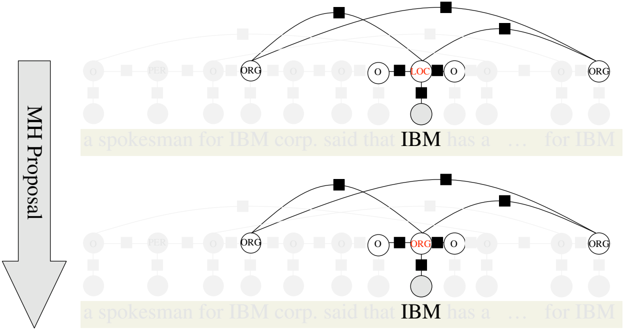


Figure: A conditional random field is a special Markov network where some nodes are *observed* (text words). Therefore, it also fits into the edge-consistency model.

Outline

Conclusion

- GraphLab is a MapReduce improvement specially designed for iterative ML tasks.
- GraphLab is good at graph iterative algorithms, e.g. PageRank.

Discussion

- How many ML tasks be modeled as graph computations?
- A guideline to pick the right tool? MapReduce, Datapath, Pregel, GraphLab, Spark, etc.
  - [Lin12] gives us both an academic and engineering point of view.
- In-memory implementation still has its limitations. How does this compare to in-database analytics?

Outline

References I

S. Arumugam, A. Dobra, C.M. Jermaine, N. Pansare, and L. Perez.
The datapath system: a data-centric analytic processing engine for large data warehouses.
In *Proceedings of the 2010 international conference on Management of data*, pages 519–530. ACM, 2010.

Y. Bu, B. Howe, M. Balazinska, and M.D. Ernst.
Haloop: Efficient iterative data processing on large clusters.
Proceedings of the VLDB Endowment, 3(1-2):285–296, 2010.

DSR @ UF
Data Science Research

References II

J. Gonzalez, Y. Low, A. Gretton, C. Guestrin, and UCL Gatsby Unit.
Parallel gibbs sampling: From colored fields to thin junction trees.
Artificial Intelligence and Statistics AISTATS, 2011.

D. Koller and N. Friedman.
Probabilistic graphical models: principles and techniques.
MIT press, 2009.

F.R. Kschischang, B.J. Frey, and H.A. Loeliger.
Factor graphs and the sum-product algorithm.
Information Theory, IEEE Transactions on, 47(2):498–519, 2001.

DSR @ UF
Data Science Research

References III

Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J.M. Hellerstein.
Distributed graphlab: a framework for machine learning and data mining in the cloud.
Proceedings of the VLDB Endowment, 5(8):716–727, 2012.

Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J.M. Hellerstein.
Graphlab: A new framework for parallel machine learning.
arXiv preprint arXiv:1006.4990, 2010.

J. Lin.
Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!
arXiv preprint arXiv:1209.2191, 2012.

DSR @ UF
Data Science Research

References IV

- G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski.
  Pregel: a system for large-scale graph processing.
  In *Proceedings of the 2010 international conference on Management of data*, pages 135–146. ACM, 2010.

- S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis.
  Dremel: interactive analysis of web-scale datasets.
  Proceedings of the VLDB Endowment, 3(1-2):330–339, 2010.

- F. Niu, C. Ré, A.H. Doan, and J. Shavlik.
  Tuffy: Scaling up statistical inference in markov logic networks using an rdbms.
  Proceedings of the VLDB Endowment, 4(6):373–384, 2011.

References V

F. Niu, C. Zhang, C. Ré, and J. Shavlik.
Felix: Scaling inference for markov logic with an operator-based approach.
arXiv preprint arXiv:1108.0294, 2011.

M. Richardson and P. Domingos.
Markov logic networks.
Machine learning, 62(1):107–136, 2006.

F. Rusu and A. Dobra.
Glade: a scalable framework for efficient analytics.
ACM SIGOPS Operating Systems Review, 46(1):12–18, 2012.

References VI

C. Sutton and A. McCallum.
An introduction to conditional random fields for relational learning.
Introduction to statistical relational learning. MIT Press, 2006.

L.G. Valiant.
A bridging model for parallel computation.
Communications of the ACM, 33(8):103–111, 1990.

M. Wick, A. McCallum, and G. Miklau.
Scalable probabilistic databases with factor graphs and mcmc.
Proceedings of the VLDB Endowment, 3(1-2):794–804, 2010.

DSR @ UF
Data Science Research

References VII

M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica.
Spark: cluster computing with working sets.
In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10. USENIX Association, 2010.