



Large-Scale Graph Processing 2: GraphLab

Shiow-yang Wu (吳秀陽)

CSIE, NDHU, Taiwan, ROC

Lecture material is mostly home-grown, partly taken with permission and courtesy from Professor Shih-Wei Liao of NTU.

GraphLab



- Open-source large graph processing system
- Implemented in C++ at CMU
- **GAS** (**G**ather, **A**pply, **S**catter) model (more on this later)
- Shared memory -> Distributed GraphLab



Main Reference

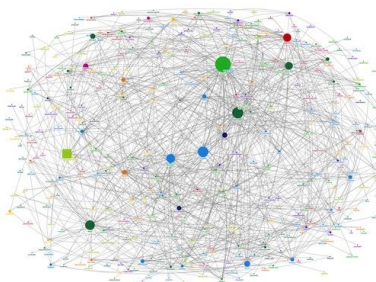


- Yucheng Low, et. al. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud, *Proceedings of the VLDB Endowment*, Vol. 5, No. 8, 2012.

Why do we need GraphLab?



- **Machine Learning and Data Mining (MLDM)** problems increasingly need systems that can execute MLDM algorithms in parallel on large clusters.
- Implementing MLDM algorithms in parallel on current systems like Hadoop and MPI can be both prohibitively complex and costly.
- The MLDM community needs a high-level abstraction to handle the complexities of graph and network algorithms.

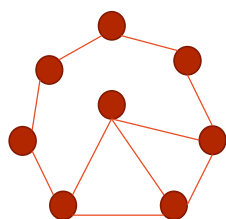


MLDM Algorithm Properties

Graph Structured Computation



Dependency Graph

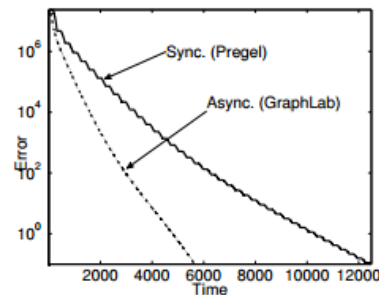


- Many of the recent advances in MLDM have focused on modeling the **dependencies** between data.
- By modeling dependencies, we are able to **extract more signal** from noisy data.

Asynchronous Iterative Computation



- **Synchronous** systems update all parameters simultaneously (in parallel) using parameter values from the previous time step as input
- **Asynchronous** systems update parameters using the most recent parameter values as input.
- Many MLDM algorithms benefit from **asynchronous** systems.

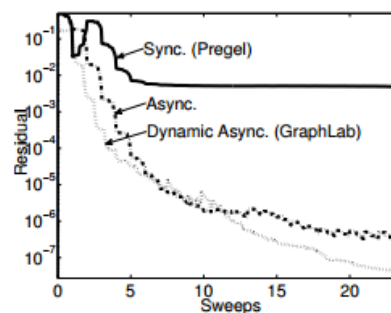


Async vs Sync PageRank

Dynamic Computation



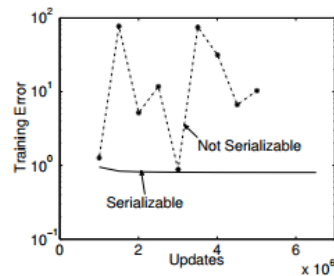
- **Static** computation requires the algorithm to update all vertices equally often. This wastes time recomputing vertices who have effectively converged.
- **Dynamic** computation allows the algorithm to potentially save time by only recomputing vertices whose neighbors have recently updated.



(c) LoopyBP Conv.

Serializability

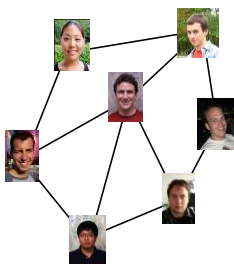
- **Serializability** ensures that all parallel executions have an equivalent sequential execution, which eliminates *race conditions*.
- Race conditions are a programming fault which can produce undetermined program states and behaviors.
- Many MLDM algorithms converge faster if serializability is ensured. Some, like Dynamic Advanced Life Support algorithm, require serializability for correctness and/or stability.



(d) ALS Consistency

Properties of MLDM Graph Processing

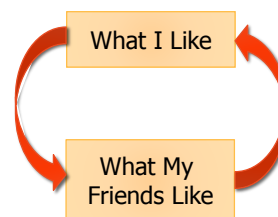
Dependency Graph



Factored Computation



Iterative Computation



GraphLab Abstraction

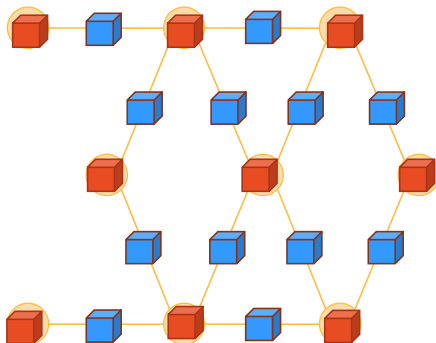


- GraphLab abstraction:
 - **Data Graph**: program state with **data** and **dependencies**
 - **Update Function**: **computation** on the data graph by transforming data in overlapping contexts called *scopes*
 - **Execution Model**: a simple **loop** execution semantics
 - **Ensuring Serializability**: models and methods to optimize parallel execution while maintaining **serializability**
 - **Sync Operation** and **Global Values**: **global values** that may be read by update functions, but are written using **sync** operations

Data Graph



- A **graph** with arbitrary data (C++ Objects) associated with each vertex and edge.



Graph: 

- Social Network

Vertex Data: 

- User profile text
- Current interests estimates

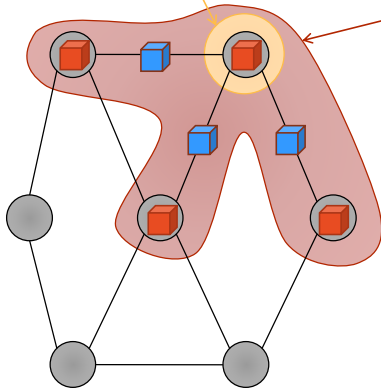
Edge Data: 

- Similarity weights

Update Functions



An **update function** is a user defined program which when applied to a **vertex** transforms the data in the **scope** of the vertex



```
label_prop(i, scope){
  // Get Neighborhood data
  (Likes[i], Wij, Likes[j]) ← scope;

  // Update the vertex data
  Likes[i] ← ∑j ∈ Friends[i] Wij × Likes[j];

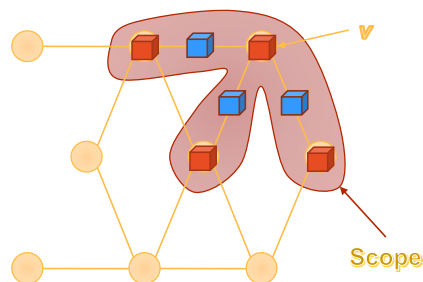
  // Reschedule Neighbors if needed
  if Likes[i] changes then
    reschedule_neighbors_of(i);
}
```

Update Functions



- An **update function** is a stateless procedure that modifies the data within the scope of a vertex and schedules the future execution of the update functions on other vertices.
- GraphLab update takes a vertex v and its scope S_v and returns the new versions of the data in the scope as well as a set vertices T :

Update: $f(v, S_v) \rightarrow (S_v, T)$



Execution Model: The Algorithm



Algorithm 2: GraphLab Execution Model

Input: Data Graph $G = (V, E, D)$

Input: Initial vertex set $\mathcal{T} = \{v_1, v_2, \dots\}$

while \mathcal{T} is not Empty **do**

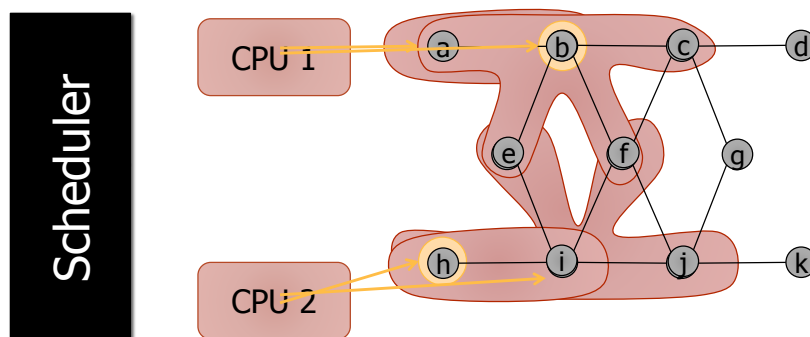
- 1 $v \leftarrow \text{RemoveNext}(\mathcal{T})$
- 2 $(\mathcal{T}', \mathcal{S}_v) \leftarrow f(v, \mathcal{S}_v)$
- 3 $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$

Output: Modified Data Graph $G = (V, E, D')$

Execution Model: The Scheduler




The **scheduler** determines the order that vertices are updated.

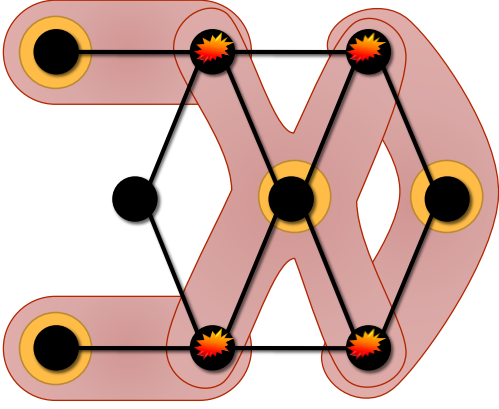


The process repeats until the scheduler is empty.

Ensuring Race-Free Code




- How much can computation **overlap**?



© Can Stock Photo - iag1524007

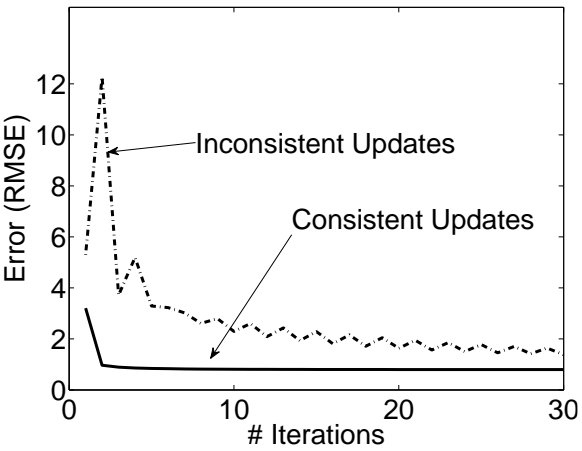
CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 18

Importance of Consistency



- Many algorithms require strict consistency, or performs significantly better under strict consistency.


Alternating Least Squares



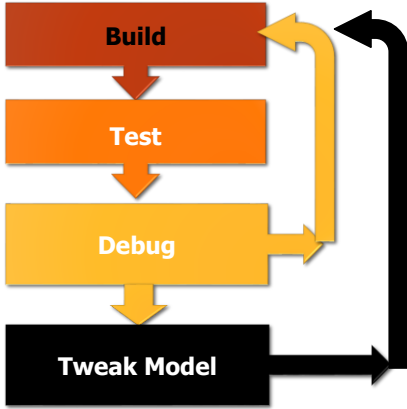
# Iterations	Error (RMSE) - Consistent Updates	Error (RMSE) - Inconsistent Updates
0	3.5	5.5
1	1.0	12.0
2	1.0	4.0
3	1.0	5.5
4	1.0	3.5
5	1.0	3.5
10	1.0	2.5
20	1.0	2.0
30	1.0	1.5

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 19

Importance of Consistency




- Machine learning algorithms require “model debugging”

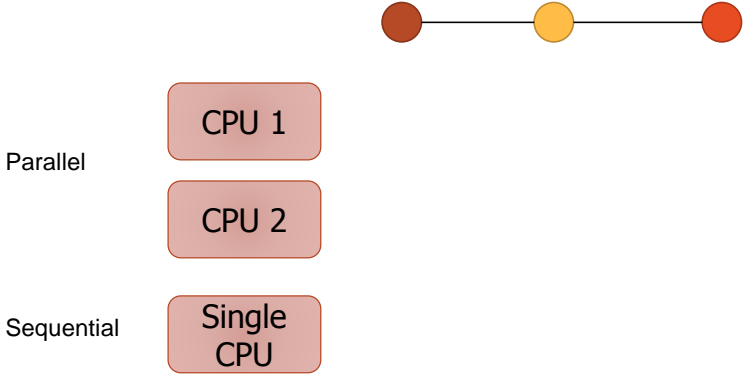


CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 20

GraphLab Ensures Sequential Consistency



For **each parallel execution**, there exists a **sequential execution** of update functions which produces the same result.



Parallel

CPU 1


CPU 2

Sequential

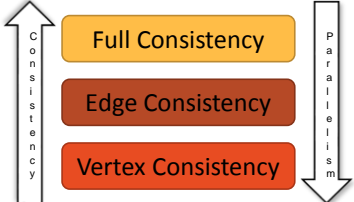
Single CPU

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 21

Ensuring Serializability




- GraphLab ensures a serializable execution by stipulating that for **every parallel execution**, there exists a **sequential execution** of update functions which produces the same result.
- GraphLab several **consistency models** which allow the runtime to optimize the parallel execution while maintaining **serializability**.
- The greater the **consistency**, the lower the **parallelism**.

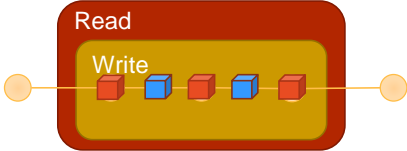


CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 23

Full Consistency




- A **full consistency** ensures that the scopes of concurrently executing update functions do not overlap.
- The **update function** has complete read-write access to its entire scope.
- This limits the potential parallelism since concurrently executing update functions must be at least **two** vertices apart.



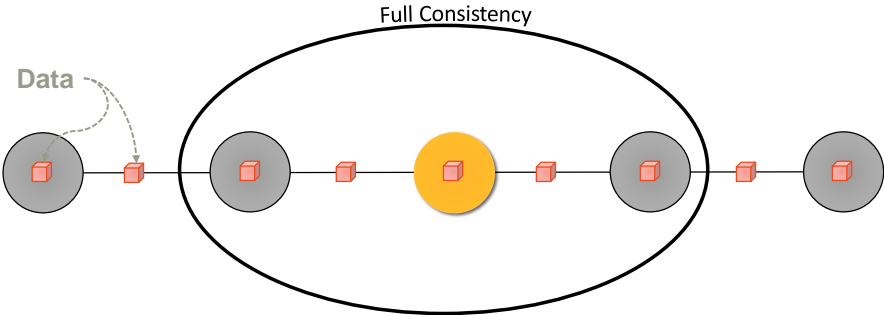
CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 24

Consistency Rules



© Can Stock Photo - 1091520407


Full Consistency



Guaranteed sequential consistency for all update functions

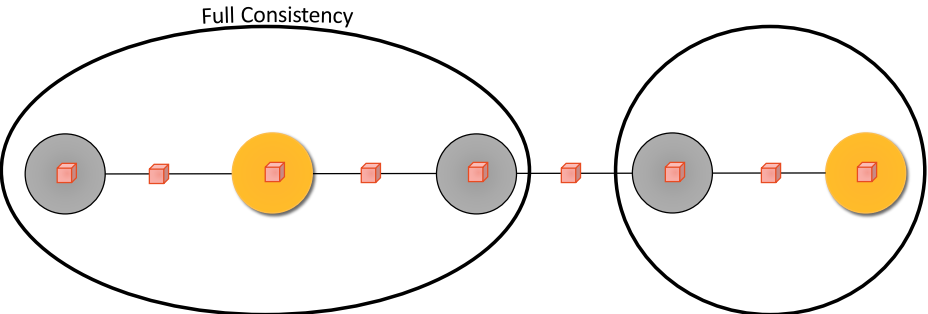
CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 25

Full Consistency



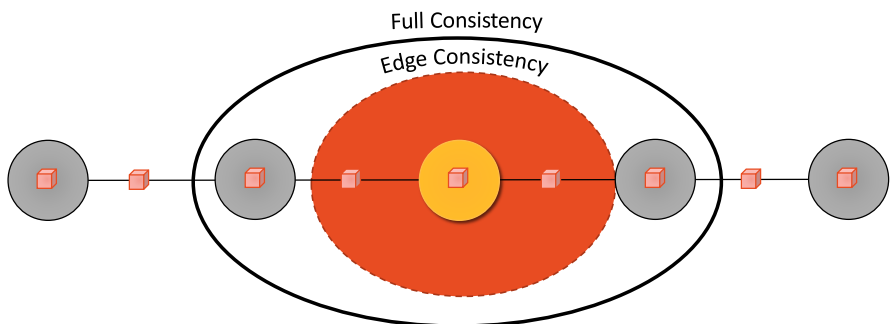
© Can Stock Photo - 1091520407

Full Consistency



CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 26

Obtaining More Parallelism

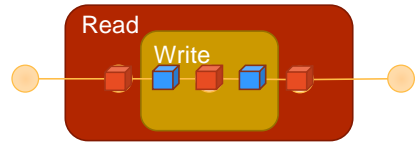


Full Consistency
Edge Consistency

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 27

Edge Consistency

- The **edge consistency** model ensures each update function has exclusive **read-write** access to its vertex and adjacent edges, but **read-only** access to adjacent vertices
- This increases parallelism by allowing update functions with slightly overlapping scopes to safely run in parallel.



Read
Write

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 28

Edge Consistency

Edge Consistency

Safe

CPU 1

CPU 2

Read

CSIE59830 Big Data Systems

Large-Scale Graph Processing 2 – GraphLab 29

Vertex Consistency

- The **vertex consistency** model only provides write access to the central vertex data.
- This allows all update functions to be run in parallel, providing *maximum parallelism*.
- However, this is the least consistent model available.



Read

Write

CSIE59830 Big Data Systems

Large-Scale Graph Processing 2 – GraphLab 30


Global Values

- Many MLDM algorithms require the maintenance of **global statistics** describing data stored in the data graph.
- GraphLab defines **global values** as values which are read by update functions and written with **sync operations**.

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 31

Sync Operation

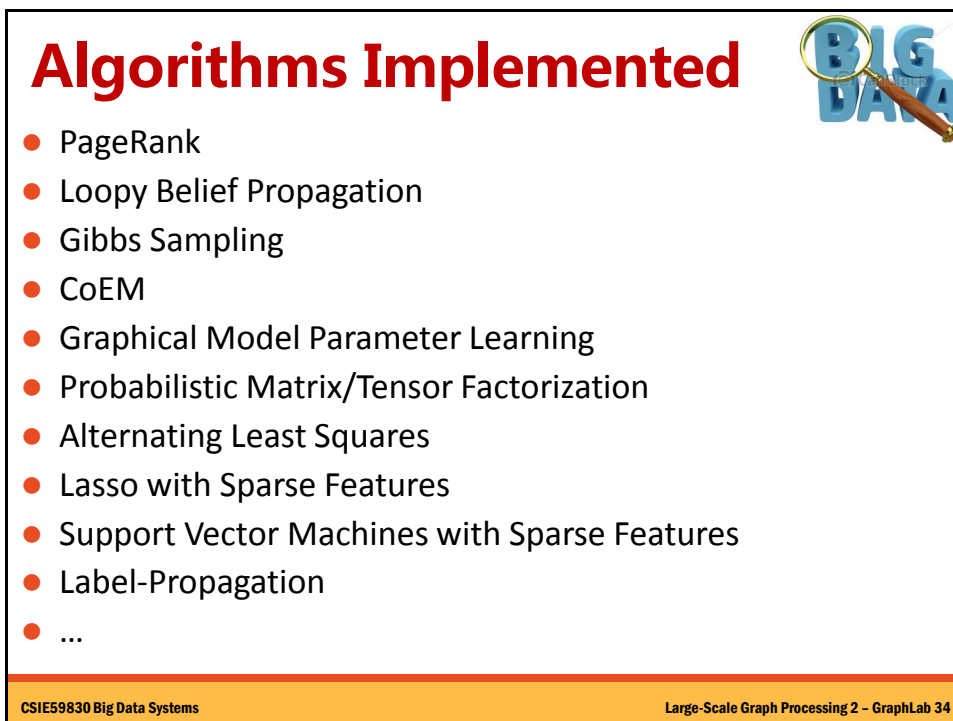
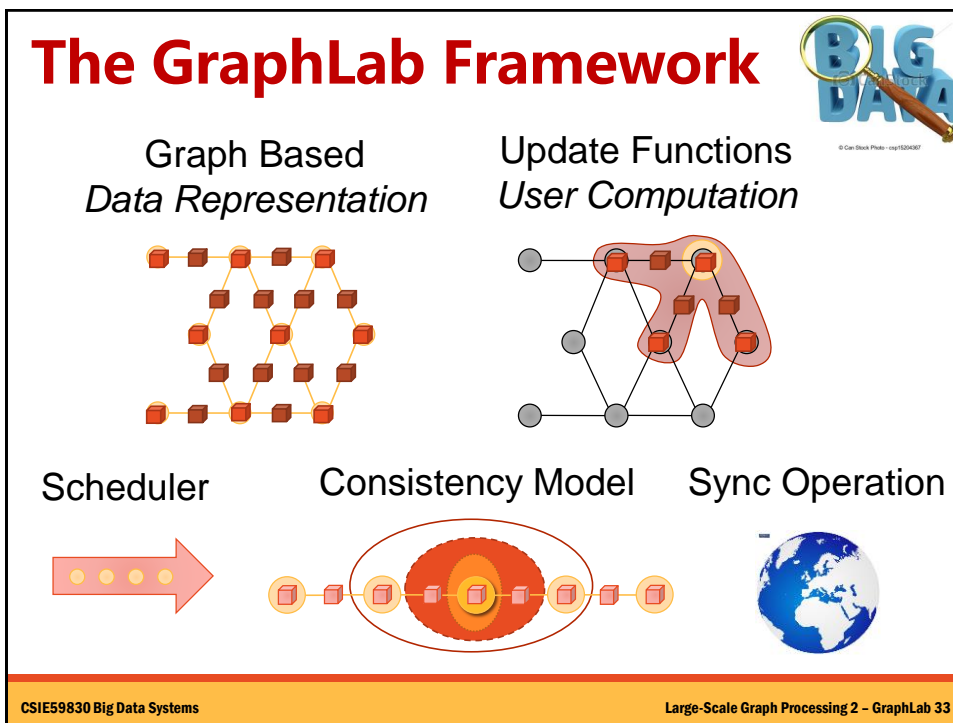


- The **sync operation** is an associative commutative sum which is defined over all parts of the graph.
- This supports tasks like normalization that are common in MLDM algorithms.
- The sync operation **runs continuously** in the **background** to maintain **updated estimates** of the global value.
- Ensuring serializability of the sync operation is costly and requires synchronization and halting all computation.

Sync Operation

$$Z = \mathbf{Finalize}(\oplus_{v \in V} \mathbf{Map}(S_v))$$

CSIE59830 Big Data Systems Large-Scale Graph Processing 2 – GraphLab 32

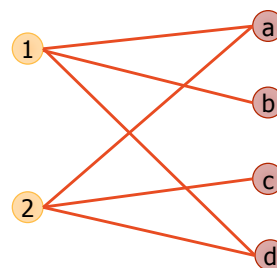


Applications


Netflix Movie Recommendation



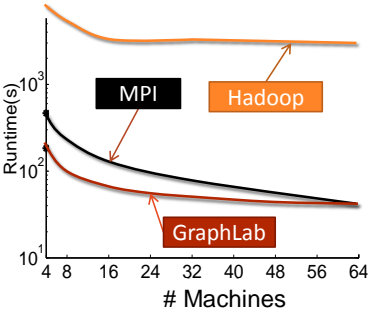
- The Netflix movie recommendation task uses **collaborative filtering** to predict the movie ratings for each user based on the ratings of similar users.
- The **alternating least squares (ALS)** algorithm is often used and can be represented using the GraphLab abstraction
- The sparse matrix R defines a bipartite graph connecting each user with the movies that they rated. Vertices are users and movies and edges contain the ratings for a user-movie pair.



Netflix Comparisons



- The GraphLab implementation was compared against Hadoop and MPI using between 4 to 64 machines.
- GraphLab performs between 40-60 times faster than Hadoop.
- It also slightly outperformed the optimized MPI implementation.



Runtime(s)

Machines

MPI


Hadoop

GraphLab


CSIE59830 Big Data Systems

Large-Scale Graph Processing 2 – GraphLab 37

Video Co-segmentation (CoSeg)



- Video co-segmentation automatically identifies and clusters spatio-temporal segments of video that share similar texture and color characteristics.
- Frames of high-resolution video are processed by coarsening each frame to a regular grid of rectangular **super-pixels**.
- The **CoSeg** algorithm predicts the best label (e.g. sky, building, grass, pavement, trees for each super pixel).



High-Res Image

Super-Pixel

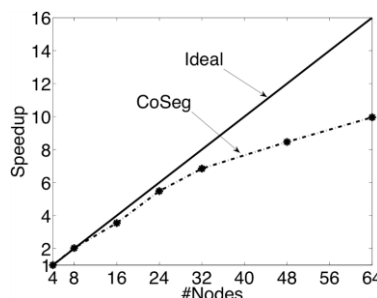
CSIE59830 Big Data Systems

Large-Scale Graph Processing 2 – GraphLab 38

CoSeg Algorithm Implementation



- CoSeg uses Gaussian Mixture Model in conjunction with Loopy Belief Propagation.
- Updates that are expected to change vertex values significantly are prioritized.
- Distributed GraphLab is the only distributed graph abstraction that allows the use of prioritized scheduling.
- CoSeg scales excellently due to having a very sparse graph and high computational intensity.



Named Entity Recognition (NER)

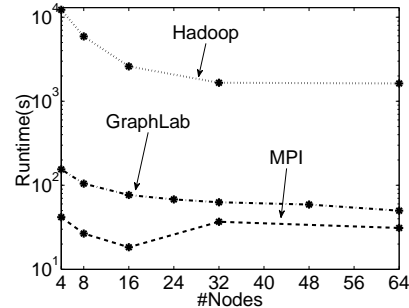


- Named Entity Recognition is the task of determining the type (e.g., *Person*, *Place*, or *Thing*) of a **noun-phrase** (e.g. *Obama*, *Chicago*, or *Car*) from its **context** (e.g. “*President..*”, “*Lives near..*”, or “*bought a..*”).
- The data graph of bipartite with one set of vertices corresponding to the noun-phrases and other corresponding to each contexts.
- There is an edge between a noun-phrase and a context if the noun-phrase occurs in the context.

NER Comparisons



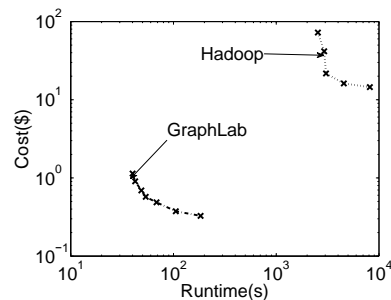
- The GraphLab implementation of NER achieved 20-30x speedup over Hadoop and was comparable to the optimized MPI.
- However, GraphLab scaled poorly achieving only a 3x improvement using 16x more machines.
- This poor performance can be attributed to the large vertex data size, dense connectivity, and poor partitioning.



Cost-Effectiveness



- The price-runtime curves for GraphLab and Hadoop illustrate the monetary cost of deploying either system.
- The price-runtime curve demonstrates diminishing returns: the cost of attaining reduced runtimes increases faster than linearly.
- For the Netflix application, GraphLab is about **two orders of magnitude** more cost-effective than Hadoop.



Summary



- An abstraction tailored to Machine Learning and Data Mining applications
 - Targets Graph-Parallel Algorithms
- Naturally expresses
 - Data/computational dependencies
 - Dynamic iterative computation
- Simplifies parallel algorithm design
- Automatically ensures data consistency
- Achieves state-of-the-art parallel performance on a variety of problems

Summary



- Distributed GraphLab extends the shared memory GraphLab to the distributed setting by:
 - Refining the execution model
 - Relaxing the schedule requirements
 - Introducing a new distributed data-graph
 - Introducing new execution engines
 - Introducing fault tolerance.
- Distributed Graphlab outperforms Hadoop by 20-60x and is competitive with tailored MPI implementations.