Datacenter Simulation Methodologies: GraphLab

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi and Benjamin C. Lee

Duke University

Tutorial Schedule

Time	**Topic**
09:00 - 10:00	Setting up MARSSx86 and DRAMSim2
10:00 - 10:15	Break
10:15 - 10:45	Web search simulation
10:45 - 11:15	**GraphLab simulation**
11:15 - 12:00	Spark simulation
12:00 - 13:00	Questions, Hands-on Session

Agenda

- Objectives
  - be able to deploy graph analytics framework
  - be able to simulate GraphLab engine, tasks
- Outline
  - Learn GraphLab for recommender, clustering
  - Instrument GraphLab for simulation
  - Create checkpoints
  - Simulate from checkpoints

Types of Graph Analysis

- Iterative, batch processing over entire graph dataset
  - Clustering
  - PageRank
  - Pattern Mining
- Real-time processing over fraction of the entire graph
  - Reachability
  - Shortest-path
  - Graph pattern matching

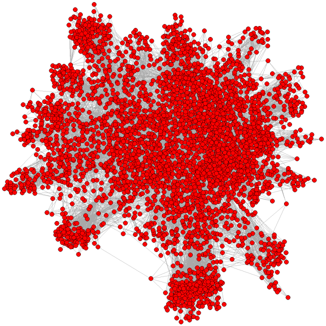
Parallel Graph Algorithms

- Common Properties
  - Sparse data dependencies
  - Local computations
  - Iterative updates
- Difficult programming models
  - Race conditions, deadlocks
  - Shared memory synchronization

"GraphLab: A New Framework for Parallel Machine Learning" by Low *et al.*

Graph Computation Challenges

- Poor memory locality
- I/O intensive
- Limited data parallelism
- Limited scalability



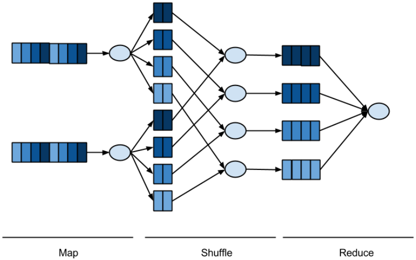
http://infolab.stanford.edu

Lumsdaine et. al. [Parallel Processing Letters 07]

MapReduce for Graphs

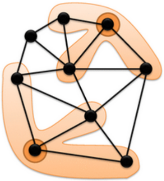
MapReduce performs poorly for parallel graph analysis

- MapReduce does not efficiently express dependent data
- Graph is re-loaded, re-processed iteratively
- MapReduce writes intermediate results to disk between iterations



GraphLab, An Alternative Approach

- Captures data dependencies
- Performs iterative analysis
- Updates data asynchronously
- Enables parallel execution models
    - Multiprocessor
    - Distributed machines

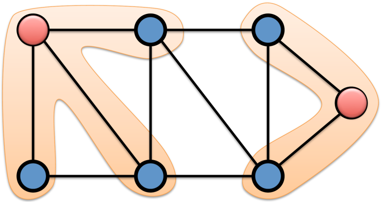


www.select.cs.cmu.edu/code/graphlab

Y. Low et. al., Distributed GraphLab, VLDB 12

The GraphLab Framework

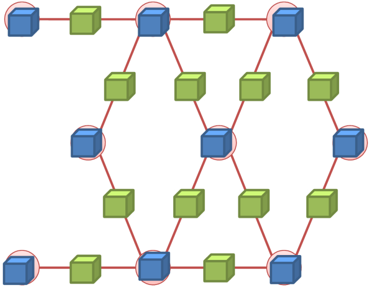
- Represent data as graph
- Specify update functions, user computation
- Choose consistency model
- Choose task scheduler



www.cs.cmu.edu/~pavlo/courses/fall2013/static/slides/graphlab.pdf

Represent Data as Graph

- Data graph associates data to each vertex and edge



Graph:
- E.g., social network

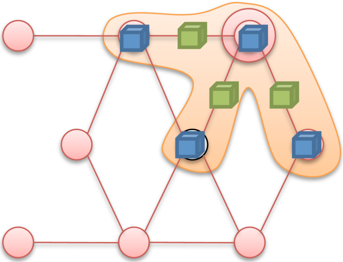
Vertex Data:
- E.g., user profile text
- E.g., interests estimates

Edge Data:
- E.g., similarity weights

C. Guestrin. A distributed abstraction for large-scale machine learning.

Update Functions and Scope

- Computation with stateless
- Scheduler prioritizes computation
- Scope determines affected edges and vertices



http://www.cs.cmu.edu/~pavlo/courses/fall2013/static/slides/graphlab.pdf

Scheduling Tasks and Updates

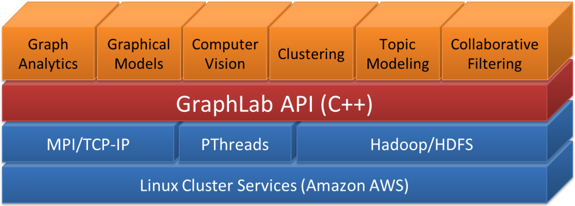
The scheduler determines the order that vertices are updated.

- Round-robin: vertices are updated in a fixed order
- FIFO: Vertices are updated in the order they are added
- Priority: Vertices are updated in priority order

Obtain different scheduling algorithms by simply changing a flag.

GraphLab Software Stack

- Collaborative filtering – recommendation system
- Clustering – Kmeans++



http://img.blog.csdn.net/

Summary

- An abstraction tailored to Machine Learning and targets Graph-Parallel Algorithms
- Naturally expresses:
  - Data/computational dependencies
  - Dynamic iterative computation
- Simplifies parallel algorithm design
- Automatically ensures data consistency

Questions?

- More information: http://graphlab.com/

Datacenter Simulation Methodologies
Getting Started with GraphLab

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi and Benjamin C. Lee

Agenda

- Objectives
  - be able to deploy graph analytics framework
  - be able to simulate GraphLab engine, tasks
- Outline
  - Learn GraphLab for recommender, clustering
  - Instrument GraphLab for simulation
  - Create checkpoints
  - Simulate from checkpoints

GraphLab Setup

- Get a product key from:
  http://graphlab.com/products/create/quick-start-guide.html
- Launch QEMU emulator:

```
$ qemu-system-x86_64 -m 4G -drive file=demo.
    qcow2,cache=unsafe -nographic
```

- In QEMU, install required tools and GraphLab-create python package

```
# apt-get install python-pip python-dev build
    -essential gcc
# pip install graphlab-create==1.1
```

GraphLab Setup

- Register product with generated key by opening file /root/.graphlab/config and editing it as follows

```
[Product]
product_key=''<generated_key>''
```

- Create a directory for GraphLab

```
# mkdir graphlab
# cd graphlab
```

- Create a directory for the dataset

```
# mkdir dataset
# cd dataset
```

Downloading a Dataset

- Download the dataset: 10 million movie ratings by 72,000 users on 10,000 movies

```
# wget files.grouplens.org/datasets/movielens
   /ml-10m.zip
# unzip ml-10m.zip
# sed 's/::/,/g' ml-10M100K/ratings.dat >
   ratings.csv
```

- Open the file and add column names on the first line: userid,moveid,rating,timestamp

GraphLab Recommender Toolkit

We will create a factorization recommender program.

- Create a new python file called recommender.py

```
import graphlab as gl
data =
gl.SFrame.read_csv('/root/graphlab/datasets/
    ratings.csv',
column_type_hints={'rating':int},header=True)
model =
gl.recommender.create(data,user_id='userid',
    item_id='movieid',target='rating')
results = model.recommend(users=None,k=5)
print results
```

- The *gl.recommender.create(args)* command chooses a recommendation model based on the input dataset format, which is the factorization recommender in this case.

GraphLab Recommender Toolkit

- The user can specify recommendation model
  - item similarity recommender,
  - factorization recommender,
  - ranking factorization recommender,
  - popularity-based recommender.
- When user specifies model explicitly, she can also specify
  - number of latent factors,
  - number of maximum iterations, etc.
- *model.recommend(args)* returns the k-highest scored items for each user. When users parameter is `None`, it returns recommendation for *all* users.

Setup for Creating Checkpoints

- Copy file *ptlcalls.h* from marss.dramsim directory

```
# scp user01@sail03.egr.duke.edu:/home/user01
   /marss.dramsim/ptlsim/tools/ptlcalls.h .
```

- Create libptlcalls.cpp file (next slide)

libptlcalls.cpp

```
#include <iostream>
#include "ptlcalls.h"
#include <stdlib.h>

extern "C" void create_checkpoint(){
  char *ch_name = getenv("CHECKPOINT_NAME");
  if(ch_name != NULL) {
    printf("creating checkpoint %s\n",ch_name);
    ptlcall_checkpoint_and_shutdown(ch_name);
  }
}

extern "C" void stop_simulation(){
  printf("Stopping simulation\n");
  ptlcall_kill();
}
```

Compile libptlcalls.cpp

- Compile C++ code

```
# g++ -c -fPIC libptlcalls.cpp -o libptlcalls
    .o
```

- Create shared library for Python

```
# g++ -shared -Wl,-soname,libptlcalls.so
    -o libptlcalls.so libptlcalls.o
```

Setup for Creating Checkpoints

- Include the library in recommender.py source code

```
from ctypes import cdll
lib = cdll.LoadLibrary('./libptlcalls.so')
```

- Call function to create checkpoint before the recommender is created. Stop the simulation after recommend function.

```
lib.create_checkpoint()
model = gl.recommender.create(data, user_id='
    userid', item_id='movieid', target='rating
    ')
lib.stop_simulation()
results = model.recommend(users=None, k=20)
```

Creating Checkpoints

- Shutdown QEMU emulator

```
# poweroff
```

- Once the emulator is shut down change into the marss.dramsim directory

```
$ cd marss.dramsim
```

- Run MARSSx86' QEMU emulator

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/demo.qcow2,cache=unsafe
    -nographic
```

- Export CHECKPOINT_NAME

```
# export CHECKPOINT_NAME=graphlab
```

- Run recommender.py

```
# python graphlab/recommender.py
```

Running from Checkpoints

- Add *-simconfig demo.simcfg* to specify the simulation configuration
- Add *-loadvm* option to load from newly created checkpoint
- Add *-snapshot* to prevent the simulation from modifying disk image

```
> ./qemu/qemu-system-x86_64 -m 4G -drive file=/
   hometemp/userXX/demo.qcow2,cache=unsafe -
   nographic -simconfig demo.simcfg -loadvm
   graphlab -snapshot
```

We will now perform k-means++ clustering

- We will use airline ontime information for 2008
- Download dataset from Statistical Computing web site. Decompress it

```
# wget stat-computing.org/dataexpo/2009/2008.
    csv.bz2
# bzip2 -d 2008.csv.bz2
```

GraphLab Clustering Toolkit

- Create a new python file called clustering.py

```
import graphlab as gl
from math import sqrt
data_url='2008.csv'
data = gl.SFrame.read_csv(data_url)
#remove empty rows
data_good, data_bad = data.dropna_split()
#determine the number of rows in the dataset
n = len(data_good)
#compute the number of clusters to create
k = int(sqrt( n / 2.0))
print "Starting k-means with %d clusters" %k
model = gl.kmeans.create(data_good,
    num_clusters=k)
##print some information on clusters created
model['cluster_info'][['cluster_id', '
    __within_distance__', '__size__']]
```

Setup for Creating Checkpoints

- Include the library in clustering.py source code

```
from ctypes import cdll
lib = cdll.LoadLibrary('./libptlcalls.so')
```

- Call the function to create checkpoint before k-means clustering model is created.

```
print "Starting k-means with %d clusters" %k
lib.create_checkpoint()
model = gl.kmeans.create(data_good,
    num_clusters=k)
lib.stop_simulation()
```

Creating Checkpoints

- Shutdown QEMU emulator

```
# poweroff
```

- Once the emulator is shut down change into the marss.dramsim directory

```
$ cd marss.dramsim
```

- Run MARSSx86' QEMU emulator

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/demo.qcow2,cache=unsafe
    -nographic
```

- Export CHECKPOINT_NAME

```
# export CHECKPOINT_NAME=kmeans
```

Running from Checkpoints

- Run clustering.py

```
# python graphlab/clustering.py
```

- The checkpoint will be created. Then the VM will shutdown
- Once the VM shuts down, update demo.simcfg to specify number of instructions to simulate *-stopinsns 1B*
- Run MARSSx86 from the checkpoint

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/demo.qcow2,cache=unsafe
    -nographic -simconfig demo.simcfg -loadvm
    kmeans -snapshot
```

Datasets

Problem	Domain Contributor	Link
Misc	Amazon Web Services public datasets	dataset
Social Graphs	Stanford Large Network Dataset (SNAP)	dataset
Social Graphs	Laboratory for Web Algorithms	dataset
Collaborative Filtering	Million Song dataset	dataset
Collaborative Filtering	Movielens dataset GroupLens	dataset
Collaborative Filtering	KDD Cup 2012 by Tencent, Inc.	dataset
Collaborative Filtering (matrix factorization based methods)	University of Florida sparse matrix collection	dataset
Classification	Airline on time performance	dataset
Classification	SF restaurants dataset	dataset

GraphLab Resources: http://graphlab.org/resources/datasets.html

Agenda

- Objectives
  - be able to deploy graph analytics framework
  - be able to simulate GraphLab engine, tasks
- Outline
  - Learn GraphLab for recommender, clustering
  - Instrument GraphLab for simulation
  - Create checkpoints
  - Simulate from checkpoints

GraphLab

For more information visit www.graphlab.org

Tutorial Schedule

Time	**Topic**
09:00 - 10:00	Setting up MARSSx86 and DRAMSim2
10:00 - 10:15	Break
10:15 - 10:45	Web search simulation
10:45 - 11:15	**GraphLab simulation**
11:15 - 12:00	Spark simulation
12:00 - 13:00	Questions, Hands-on Session