Scalable Data Structures for Machine Learning

Carlos Guestrin
University of Washington
Dato, Inc.

http://www.istc-cc.cmu.edu/
The GraphLab journey

- Distributed system for graph computation
- Challenges scaling to huge graphs

Triangle Counting on Twitter Graph
40M Users
1.2B Edges
Total: 34.8 billion triangles

GraphLab2
1636 Machines, 423 Minutes
59 Minutes, 1 Mac Mini!
64 Machines, 1024 Cores
1.5 Minutes

GraphChi
- Designed for best single machine scaling
- Optimized out-of-core computation
Many systems

- Assume unbounded resources
- Optimize for scale

- Limited scalability on single machine / small cluster
- Really painful to build intelligent applications…
SFrame: Scalable data frame for ML
Data frames

When you choose a data frame, have your application in mind

SFrame is optimized for ML

ML has specific data access patterns, we make them fast, really fast
(Columnar transformations, creating new features, iterations, ...)

user    movie    rating

[Diagram of data frame with columns user, movie, rating]
Data is usually rows…

But, data engineering typically column transformations…
Feature engineering is columnar

Normalizes the feature $x$:
\[
sf[\text{\textquoteleft\textquoteleft rating\textquoteleft\textquoteleft}] = sf[\text{\textquoteleft\textquoteleft rating\textquoteleft\textquoteleft}] / sf[\text{\textquoteleft\textquoteleft rating\textquoteleft\textquoteleft}].sum()
\]

Sequential operations happen over one or a few columns, not rows of data
(certain algorithms, e.g., SGD, operate or rows, won’t cover today, but can be addressed in framework)
### Opportunity for Out-of-Core ML

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 TB</td>
<td>0.1 GB/s</td>
</tr>
<tr>
<td>1 TB</td>
<td>0.5 GB/s</td>
</tr>
<tr>
<td>0.1 TB</td>
<td>1 GB/s</td>
</tr>
</tbody>
</table>

Opportunity for big data on 1 machine

- Fast, but significantly limits data size

For sequential reads only!
Random access very slow

**GraphChi** early example
**SFrame** data frame for ML

- Out-of-core ML opportunity is huge
- Usual design → Lots of random access → Slow
- Design to maximize sequential access for ML algo patterns
SFrame: Scalable data frame optimized for ML

Never run out of memory
Sharded, compressed, out-of-core, columnar
Arbitrary lambda transformations, joins,… from Python

Large data on one machine?
Limited RAM ➔ Must use disk
(out-of-core computation)
SFrame columnar encoding

Netflix Dataset,
99M rows, 3 columns, ints
1.4GB raw
289MB gzip compressed

User → 176 MB 14.2 bits/int
Movie → 257 KB 0.02 bits/int
Rating → 47 MB 3.8 bits/int
Total → 223MB

Type aware compression:
• Variable Bit length Encode
• Frame Of Reference Encode
• ZigZag Encode
• Delta / Delta ZigZag Encode
• Dictionary Encode
• General Purpose LZ4
Demo: 10TBs of data on one machine!
SFrame ❤️ all ML
scikit-learn is awesome, but...

Out of RAM
Numpy in memory only

Airline Delay Dataset, SGDLinearClassifier

Runtime (s)
Demo: 10TBs of data on one machine *redux*
Numpy Automatically Backed by Sframes → Scale many Python packages (scikit-learn, scipy,…)

```
import graphlab.numpy
Scalable numpy activation successful
```
ML is not just about tables
• **Graphs** encode the **relationships** between:

  People  Products  Ideas
  Facts  Interests

• **Big**: trillions of vertices and edges and rich metadata
  - Facebook (10/2012): 1B users, 144B friendships
  - Twitter (2011): 15B follower edges
For example…
Example: Estimating political bias
But, ML is about all data types…
ML pipelines combine multiple data types

Raw Wikipedia

Text Table

Hyperlinks

PageRank

Top 20 Pages

Term-Doc Graph

Topic Model (LDA)

Word Topics
Integrating tables and graphs
SGraph

Graph processing & analytics

Backed by SFrame

Out-of-core & scalable

Neighborhoods, paths, graph algos, community detection, label propagation, ML on graphs, viz, …
Basic graph representation

**Vertex Table**

<table>
<thead>
<tr>
<th>__id</th>
<th>Address</th>
<th>ZipCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>…</td>
<td>98105</td>
</tr>
<tr>
<td>Bob</td>
<td>…</td>
<td>98102</td>
</tr>
<tr>
<td>Charlie</td>
<td>…</td>
<td>98103</td>
</tr>
</tbody>
</table>

**Edge Table**

<table>
<thead>
<tr>
<th>__src_id</th>
<th>__dst_id</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>“hello”</td>
</tr>
<tr>
<td>Bob</td>
<td>Charlie</td>
<td>“world”</td>
</tr>
<tr>
<td>Charlie</td>
<td>Alice</td>
<td>“moof”</td>
</tr>
</tbody>
</table>
SGraph vertex data layout

Example: vertices partitioned into $p = 4$ SFrames

<table>
<thead>
<tr>
<th>__id</th>
<th>Name</th>
<th>Address</th>
<th>ZipCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011</td>
<td>John</td>
<td>...</td>
<td>98105</td>
</tr>
<tr>
<td>2131</td>
<td>Jack</td>
<td>...</td>
<td>98102</td>
</tr>
</tbody>
</table>
SGraph edge data layout

Vertex SFrames

| 1 | 2 | 3 | 4 |

Edge SFrames

<table>
<thead>
<tr>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(1,3)</th>
<th>(1,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,1)</td>
<td>(2,2)</td>
<td>(2,3)</td>
<td>(2,4)</td>
</tr>
<tr>
<td>(3,1)</td>
<td>(3,2)</td>
<td>(3,3)</td>
<td>(3,4)</td>
</tr>
<tr>
<td>(4,1)</td>
<td>(4,2)</td>
<td>(4,3)</td>
<td>(4,4)</td>
</tr>
</tbody>
</table>

Edges partitioned into $p^2 = 16$ SFrames

<table>
<thead>
<tr>
<th>src_id</th>
<th>dst_id</th>
<th>NumLikes</th>
<th>NumMsgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>4131</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>2023</td>
<td>4234</td>
<td>23</td>
<td>3</td>
</tr>
</tbody>
</table>

Vertex SFrames:

- Vertex 1
- Vertex 2
- Vertex 3
- Vertex 4

Edge SFrames:

- Edge (1,1)
- Edge (1,2)
- Edge (1,3)
- Edge (1,4)
- Edge (2,1)
- Edge (2,2)
- Edge (2,3)
- Edge (2,4)
- Edge (3,1)
- Edge (3,2)
- Edge (3,3)
- Edge (3,4)
- Edge (4,1)
- Edge (4,2)
- Edge (4,3)
- Edge (4,4)

Edges partitioned into $p^2 = 16$ SFrames.
Deep integration of SFrames and SGraphs

• Seamless interaction between graph data and table data

**SGraph vertex data can be viewed as tables**

```python
g = SGraph(...)  
g.vertices[‘large Pagerank’] = g.vertices[‘Pagerank’]>100
```

**SGraph edge data can be viewed as tables**

```python
g = SGraph(...)  
g.edges[‘normalized_ratings’] = g.edges[‘ratings’]/g.edges[‘ratings’].mean()
```
SGraph columnar ➔ Selection is easy
SGraph columnar ➔ Adding features is easy

Vertex SFrames

Edge SFrames
Performing computations on graphs
Distributed connected components algorithm

- Initialize: Assign vertex id as component
- Iterate:
  - My id is the minimum of my neighborhood
Properties of graph-parallel algorithms

Dependency Graph

Local Updates

Iterative Computation

Graphical Models
- Gibbs Sampling
- Belief Propagation
- Variational Inf.

Collaborative Filtering
- Item-item similarity
- Tensor Factorization

Semi-Supervised Learning
- Label Propagation
- CoEM

Data-Mining
- PageRank
- Triangle Counting
Graph-parallel programming abstractions
Vertex programs [Low et al. ‘10]

User-defined program: applied to vertex transforms data in scope of vertex

```plaintext
connected_components(vertex, neighbors){
  // Compute minimum component
  min_component = min(vertex['component'],
                       components of neighbors)
  // Update vertex component
  vertex['component'] = min_component
}
```

Vertex programs simple, but exhibit significant performance challenges in out-of-core & distributed settings
**Triple_apply:**
simple, highly-parallelizable graph processing abstraction

**Edge programs** not vertex programs!

Distributed implementation is much simpler & more efficient than vertex programs

```plaintext
connected_comp_triple_apply(src, dst, edge){
    // Compute minimum component
    min_component = min(src['component'], dst['component'])
    // Update both vertices
    src['component'] = min_component
dst['component'] = min_component
}
```
Optimizing triple_apply execution
Triple_apply needs vertex data for src and dst vertices

<table>
<thead>
<tr>
<th>Vertex SFrames</th>
<th>Edge SFrames</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1) (1,2)</td>
</tr>
<tr>
<td>2</td>
<td>(2,1) (2,2)</td>
</tr>
<tr>
<td>3</td>
<td>(3,1) (3,2)</td>
</tr>
<tr>
<td>4</td>
<td>(4,1) (4,2)</td>
</tr>
</tbody>
</table>
Naïve traversal over edges

Vertex SFrames

| 1 | 2 | 3 | 4 |

Edge SFrames

<table>
<thead>
<tr>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(1,3)</th>
<th>(1,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4,1)</td>
<td>(4,2)</td>
<td>(4,3)</td>
<td>(4,4)</td>
</tr>
</tbody>
</table>

Significant IO cost
No cache locality
Need walk ordering minimizing loading-unloading

• Efficient option: Hilbert space-filling curves
  - Minimum loads of vertex data
  - Preserves locality $\Rightarrow$ great cache behavior
SGraph: performance
Performance of SGraph

Connected components in Twitter graph

- SGraph: 70 sec
- GraphX: 251 sec
- Giraph: 200 sec
- Spark: 2,128 sec

Source(s): Gonzalez et. al. (OSDI 2014)

Twitter: 41 million Nodes, 1.4 billion Edges
Pagerank on Common Crawl Graph
3.5 billion Nodes and 128 billion Edges

Minutes per iteration

1 machine
16 CPUs, 1 SSD
SFrame/SGraph Summary
**SFrame & SGraph**

**Optimized out-of-core computation for ML**

**High Performance**
- 1 machine can handle:
  - TBs of data
  - 100s Billions of edges

**Optimized for ML**
- Columnar transformation
- Create features
- Iterators
- Filter, join, group-by, aggregate
- User-defined functions
- Easily extended through SDK

**Open-source**

**BSD license**

**More than 10,000 downloads**

**Tables, graphs, text, images**

**python™ R**
A tech-transfer update…
(Not ISTC IP)
The ML pipeline circa 2013

Data → ML Algorithm → My curve is better than your curve → Write a paper
Disruptive companies differentiated by INTELLIGENT APPLICATIONS using Machine Learning
In 5 years, every successful app will be intelligent

Promise will only come to be if we change how ML is done

Today: need data scientists, who write production code & know about deployment
Very rare: thus huge investments & teams @Google, Facebook, Microsoft, Amazon
Dato’s mission is to accelerate the creation of intelligent applications by making sophisticated machine learning as easy as “Hello world!”
Demo of an intelligent application made easy
Since last year...
Sophisticated machine learning made easy
Create Intelligence Accelerants

High-level ML toolkits

AutoML
tune params, model selection,…
so you can focus on creative parts

Reusable features
transferrable feature engineering
accuracy with less data & less effort
High-level ML toolkits get started with 4 lines of code, then modify, blend, add yours…

import graphlab as gl
data = gl.SFrame.read_csv('my_data.csv')
model = gl.recommender.create(data,
    user_id='user',
    item_id='movie',
    target='rating')
recommendations = model.recommend(k=5)
GraphLab Create includes easy to use, deep learning on multi-GPUs

Deep learning in 1 line of code

You can also open the box and add your own layers

Deep learning tutorial tomorrow, 4pm!

```
graphlab.deeplearning.create(data,target=label')
```
Digit recognition benchmark

H2O.ai: 10 machines/80 cores

GraphLab Create: 4 min on 4 GPUs
Sophisticated machine learning made distributed

Create Intelligence on Huge Data

Distributed machine learning

Your big data infrastructure (cloud, hadoop, spark,..)
Pagerank on Common Crawl Graph
3.5 billion Nodes and 128 billion Edges

- 45 secs/iteration
- 3B edges/sec

Minutes per iteration

1 machine
16 CPUs
Criteo Terabyte Click Prediction

4.4 Billion Rows
13 Features

½ TB of data

![Graph showing the relationship between number of machines and runtime.]
Same code, distributed ML

```python
import graphlab as gl

data = gl.SFrame.read_csv('s3://...')

model = gl.classifier.create(data, target='click')

# Single machine ML code

c = gl.deploy.ec2_cluster.load('s3://...')

gl.set_distributed_execution_environment(c)

# ML code

c = gl.deploy.hadoop_cluster.load('hdfs://...')
c = gl.deploy.spark_cluster.load('hdfs://...')
```
SFrame/Sgraph summary
SFrame & SGraph

- Optimized out-of-core computation for ML
- Open-source BSD license
- More than 10,000 downloads

High Performance
- 1 machine can handle:
  - TBs of data
  - 100s Billions of edges

Optimized for ML
- Columnar transformation
- Create features
- Iterators
- Filter, join, group-by, aggregate
- User-defined functions
- Easily extended through SDK