TuX²: Distributed Graph Computation for Machine Learning

Wencong Xiao*, Jilong Xue†, Youshan Miao◊, Zhen Li*, Cheng Chen◊, Ming Wu◊, Wei Li*, Lidong Zhou◊

*SKLSDE Lab, Beihang University; ◊Microsoft Research; †Peking University
Machine Learning (ML) in real world

Recommendation

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>5</th>
<th>2.94*</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2.48*</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1.12*</td>
</tr>
</tbody>
</table>

topics

gene 0.87
heritability 0.86
... life 0.84
... disease 0.84
... organ 0.84
... reasoning 0.84
... strategy 0.84
... data 0.84
... computer 0.84

Documents

Seeking Life's Bare (Genetic) Necessities

Topic proportions and assignments

Technology

Sports

Entertainment
Graph Structures in Machine Learning
Advantages of Graph Engine

Simple programming model (e.g. GAS)
- PageRank, Shortest path, etc.

Graph-aware optimization
- Data layout [Grace(ATC’12), Naiad(SOSP’13)]
- Partitioning [PowerLyra(EuroSys’15)]

Scalability to trillion-edge
- GraM (SoCC’15)
- Chaos (SOSP’15)
- One Trillion Edges (VLDB’15)

*GAS figure from PowerLyra slides
Gaps for ML on Graph Engine

1. Heterogeneous vertices

PageRank for WebPage Ranking

Matrix Factorization (MF) for Recommendation
Gaps for ML on Graph Engine

2. Mini-Batch

PageRank for WebPage Ranking

Matrix Factorization(MF) for Recommendation
Gaps for ML on Graph Engine

3. Flexible consistency

PageRank for WebPage Ranking

Hard Barrier

Matrix Factorization (MF) for Recommendation
Gaps for ML on Graph Engine

3. Flexible consistency
We propose: $\text{TuX}^2$

Bridge Graph and ML research in one system

Extend for distributed machine learning
- Scheduling: Stale Synchronous Parallel (SSP) based scheduling
- DataModel: Heterogeneous data model
- Programming: MEGA (Mini-batch, Exchange, GlobalSync, and Apply) graph model

Outperform both Graph and ML systems on ML algorithms
- 10x vs. PowerGraph/PowerLyra
  - Mainly due to MEGA model and heterogeneity optimization
- 48% vs. Petuum/Parameter-Server(P-S)
  - Mainly due to graph-based optimization
System Architecture

Overview

Partition 0
Server role

... $V_i$ $V_j$ ...

Vertex array

... $V_i$ ...

Edge array

Worker role

Partition n

Master vertices

... ○ ○ ○ ...

Mirror vertices

... $V_i$ ...

Edge link

Master-mirror link

Vertex-cut approach
- Effective for power-law graph
- Naturally fits P-S model
  - Master vertices as the global state
  - Mirror vertices as the local cache
Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

DataModel: Heterogeneous data model

Programming: MEGA graph model
Stale Synchronous Parallel in TuX$^2$

Slack of 1 clock as an example

- All servers finish clock 1

Updates guaranteed visible to all servers

Working set of current mini-batch
Stale Synchronous Parallel in TuX$^2$

Slack of 1 clock as an example
- Slowest server (n) is in clock2
- Fastest server (0) finishes clock2
Stale Synchronous Parallel in TuX\textsuperscript{2}

Slack of 1 clock as an example

- Slowest server (n) is in clock2
- Fastest server (0) finishes clock2
  - within the staleness bound
  - continue
Stale Synchronous Parallel in TuX$^2$

Slack of 1 clock as an example
- Slowest server (n) is in clock2
- Fastest server (0) finishes clock3
  - reaching the max slack bound
  - blocked

Server 0 is blocked
Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

DataModel: Heterogeneous data model

Programming: MEGA graph model
Heterogeneity in ML

Heterogeneous Vertices
- Different properties
  - E.g. Logistic Regression
    - Sample: Label; Feature: Weight, Gradient

Benefit
- Heterogeneity for compact data structure
- Heterogeneity for efficient execution
- Heterogeneity for less network traffic
Heterogeneity for compact data structure

E.g. Logistic Regression
- Sample: Label; Feature: Weight, Gradient
Heterogeneity for efficient execution

E.g. Mini-Batch MF for recommendation
- Benefits of scanning items
  - Sequential access for locality when syncing
  - Less overhead tracing the updated vertices
Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

DataModel: Heterogeneous data model

Programming: MEGA graph model
**MEGA: e.g. Mini-batch MF for recommendation**

**Model:** $\hat{R}_{i,j} = I_j^T U_i$

**Graph View**

```
Exchange(v_user, v_item, edge, a_user, a_item, context)
  pred = PredictRating(v_user, v_item);
  loss = edge.rating - pred;
  context.loss += loss^2;
  (a_user, a_item) += Gradient(loss, v_user, v_item);

Apply(ver, accum, context)
  ver.data += accum;
```
Example: Mini-batch MF

Compose stage

Mini-Batch

Exchange

Apply

GlobalSync

StageSequenceBuilder(ExecStages)
mbStage = new MiniBatchStage();
mbStage.SetBatchSize(100, asEdge);
mbStage.Add(ExchangeStage);
mbStage.Add(ApplyStage);
ExecStages.Add(mbStage);
ExecStages.Add(GlobalSyncStage);
Experiment setup

Machine information
- 16 CPU cores, 256GB memory, 54Gbps InfiniBand NIC

Typical ML algorithms
- MF, LDA, BlockPG

Large-scale dataset
- Up to 64 billion edges graph

<table>
<thead>
<tr>
<th>Dataset name</th>
<th># of users/docs/samples</th>
<th># of items/words/features</th>
<th># of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewsData(LDA)</td>
<td>7.3M</td>
<td>418.4K</td>
<td>1.4B</td>
</tr>
<tr>
<td>AdsData(BlockPG)</td>
<td>924.8M</td>
<td>209.3M</td>
<td>64.9B</td>
</tr>
<tr>
<td>Netflix(MF)</td>
<td>480.2K</td>
<td>17.8K</td>
<td>100.5M</td>
</tr>
<tr>
<td>Synthesized(MF)</td>
<td>30M</td>
<td>1M</td>
<td>6.3M</td>
</tr>
</tbody>
</table>
Evaluation

Compare to Parameter Server
- 48% improvement on 32 servers!
- Algorithm: BlockPG
- Dataset: Microsoft private AdsData (64B edges)

Balance workload with vertex-cut!
Evaluation

Compare to PowerGraph, PowerLyra
- Algorithm: Matrix Factorization
- Dataset: Netflix
Evaluation

Compare to PowerGraph, PowerLyra
- Algorithm: Matrix Factorization
- Dataset: Netflix
Conclusion

TuX²: advocates the convergence of graph computation and distributed machine learning

- Introduce important machine learning concepts to graph computation
- Define a new, flexible graph model to express ML algorithms efficiently
- Demonstrate TuX² outperform existing Graph and ML systems in representative ML algorithms respectively
Thanks!
Q&A