PETUUM

a New Platform for Distributed Machine Learning on Big Data

- Seminar on Big Data Architectures for Machine Learning and Data Mining -

Bauhaus-Universität Weimar

Viorel Morari, 115629

Outline

1

Introduction

- the need
- current problems

2

Motivation

- why build a new framework?!
- scalability

3

Components & Approach

- key concepts
- data-parallelism
- model-parallelism

4

System Design

- internals - schematic view
- scheduler, workers

5

YARN compatibility

- problems & solutions proposed

6

Performance

- Comparison with different platforms

7

Example

- program structure
- simple implementation example

PETUUM

Introduction

- Machine Learning is becoming a primary mechanism for extracting information from data.
- Need ML methods to scale beyond single machine.
- Flickr, Instagram and Facebook → 10s of billions of images.
- Highly inefficient to use such big data sequentially in a batch fashion in a typical iterative ML algorithm
- Despite rapid development of many new ML models and algorithms aiming at scalable application, adoption of these technologies remains generally unseen in the wider data mining, NLP, vision, and other application communities.
- Difficult migration from an academic implementation (small desktop PCs, small lab clusters) to a big, less predictable platform (cloud or a corporate cluster) prevents ML models and algorithms from being widely applied.

PETUUM

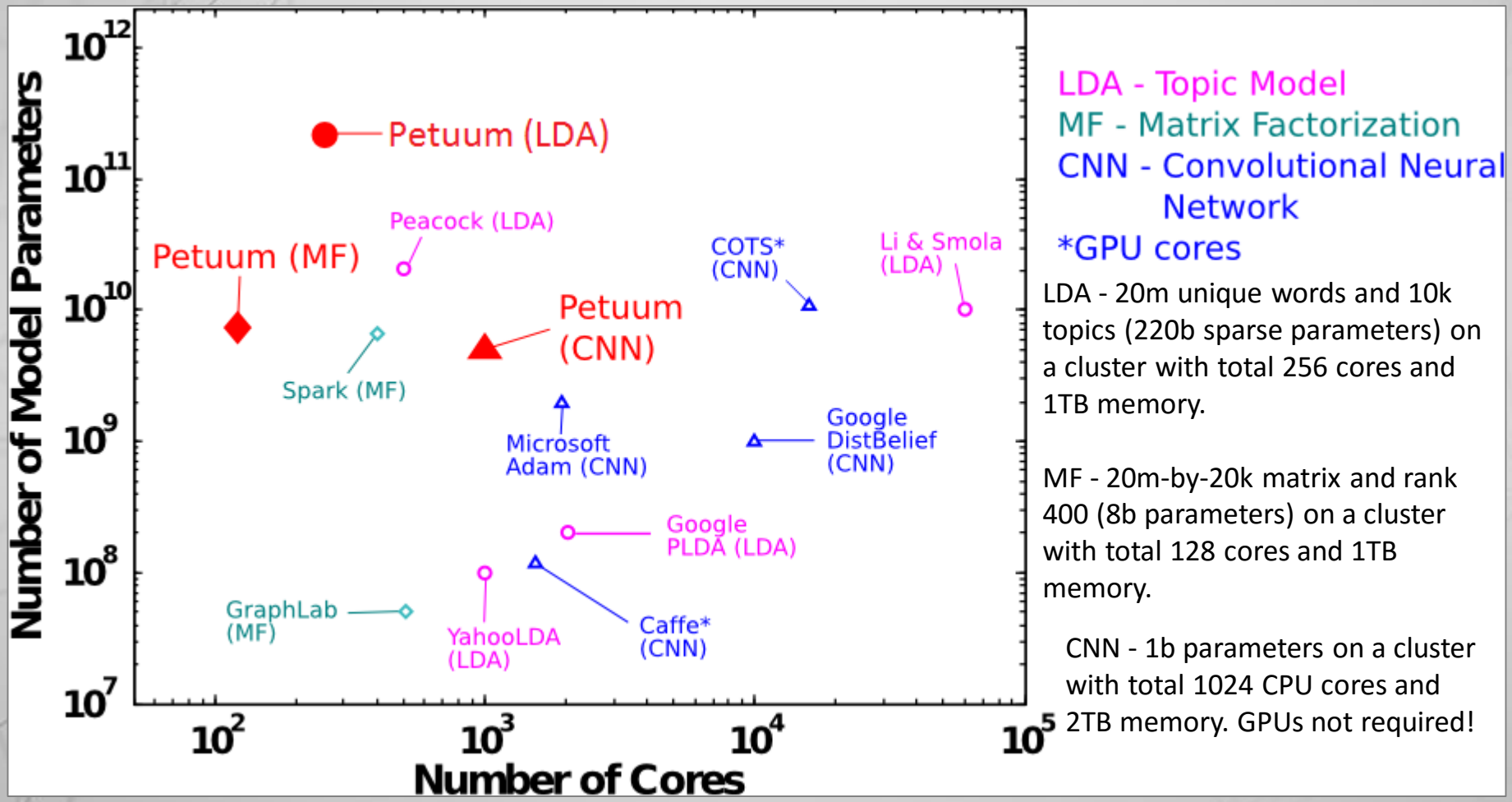
Motivation

- Find a systematic way to efficiently apply a wide spectrum of advanced ML programs to industrial scale problems, using Big Models (up to 100s of billions of parameters) on Big Data (up to terabytes or petabytes).
- Modern parallelization strategies employ fine-grained operations but it remains difficult to find a universal platform applicable to a wide range of ML programs at scale.

Problems with other platforms

- **Hadoop** :
  - simplicity of MapReduce → difficult to exploit ML properties (error tolerance)
  - performance on many ML programs has been surpassed by alternatives.
- **Spark** :
  - does not offer fine-grained scheduling of computation and communication for fast and correct execution of advanced ML algorithms.

Motivation



LDA - 20m unique words and 10k topics (220b sparse parameters) on a cluster with total 256 cores and 1TB memory.

MF - 20m-by-20k matrix and rank 400 (8b parameters) on a cluster with total 128 cores and 1TB memory.

CNN - 1b parameters on a cluster with total 1024 CPU cores and 2TB memory. GPUs not required!

Fig. 1: The scale of Big ML efforts in recent literature. A key goal of Petuum is to enable larger ML models to be run on fewer resources, even relative to highly-specialized implementations.

PETUUM

Components & Approach

- Formalized ML algorithms as iterative-convergent programs:
  - stochastic gradient descent
  - MCMC for determining point estimates in latent variable models
  - coordinate descent, variational methods for graphical methods
  - proximal optimization for structured sparsity problems, and others
- Found out the shared properties across all algorithms.
- Key lies in the recognition of a clear dichotomy(division) b/w DATA and MODEL
- This inspired bimodal approach to parallelism: data parallel and modal parallel distribution and execution of a big ML program over cluster of machines.

Components & Approach

This approach exploits unique statistical nature of ML algorithms, mainly three properties:

- **Error tolerance** – iterative-convergent algorithms are robust against limited errors in intermediate calculations.
- **Dynamic structural dependency** – changing correlation strengths between model parameters critical to efficient parallelization.
- **Non-uniform convergence** – No. of steps required for a parameter to converge can be highly skewed across parameters.

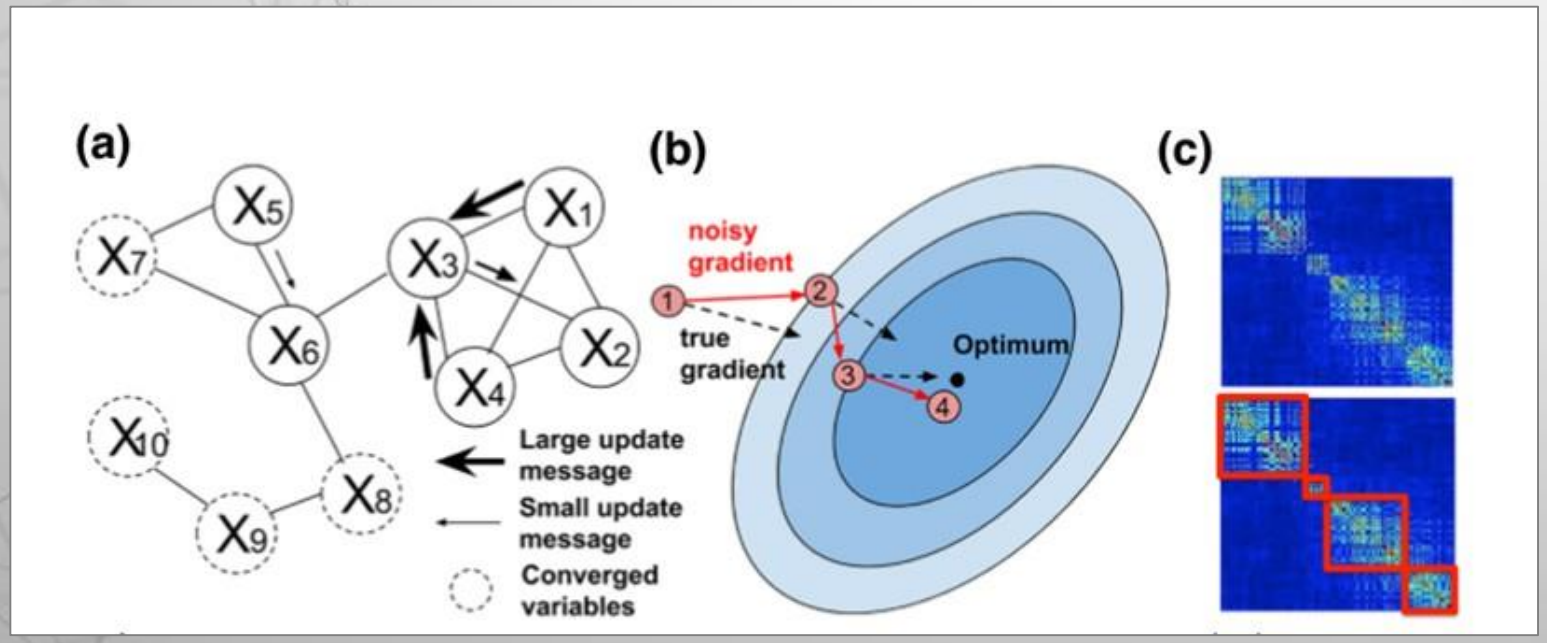


Fig. 2: Key properties of ML algorithms:
(a) Non-uniform convergence;
(b) Error-tolerant convergence;
(c) Dependency structures amongst variables.

PETUUM

Components & Approach

Iterative-Convergent ML Algorithm: Given data *D* and loss *L* (i.e., fitness function such as likelihood), a typical ML problem can be grounded as executing the following update equation iteratively, until the model state (i.e., parameters and/or latent variables)
A reaches some stopping criteria:

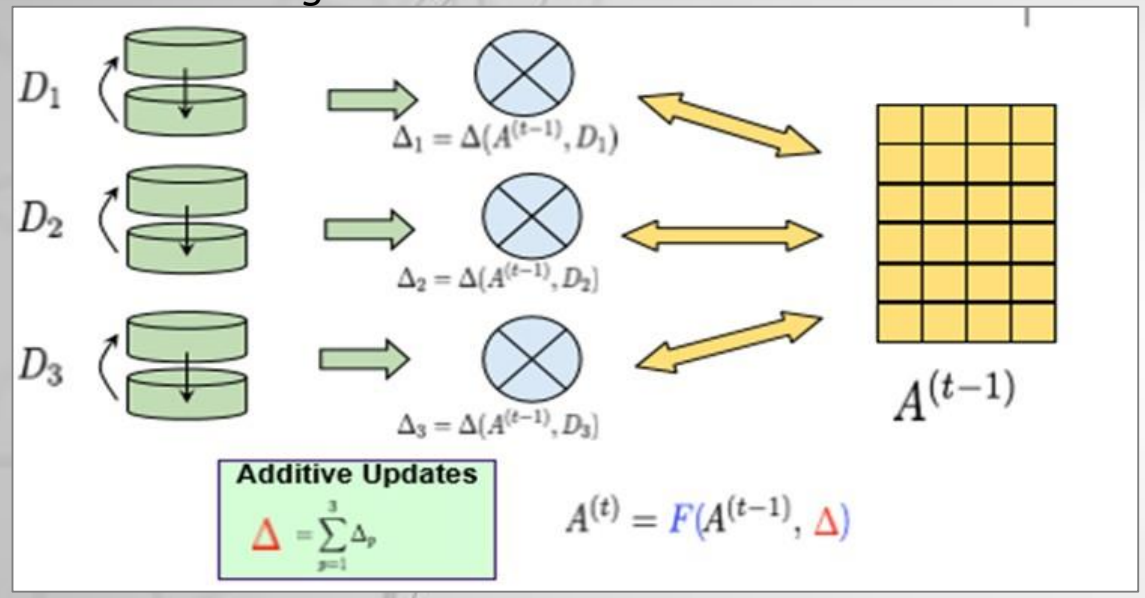
$$A^{(t)} = F(A^{(t-1)}, \Delta_{\mathcal{L}}(A^{(t-1)}, D))$$

The update function ΔL() (which improves the loss *L*) performs computation on data *D* and model state *A*, and outputs intermediate results to be aggregated by *F*().

PETUUM

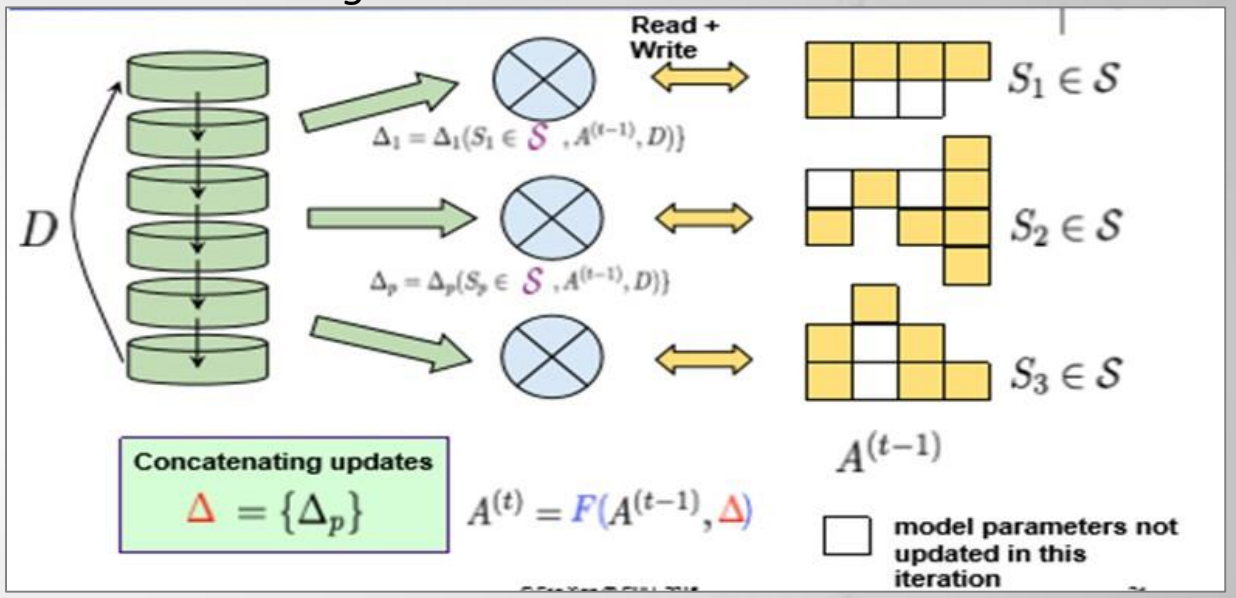
Components & Approach

Fig. 3: Petuum Data-Parallelism



- the data *D* is partitioned and assigned to computational workers;
- we assume that the function Δ() can be applied to each of these data subsets independently
- Δ() are aggregated via summation
- Each parallel worker contributes equally

Fig. 4: Petuum Model-Parallelism



- the data *A* is partitioned and assigned to workers;
- unlike data-parallelism each update function Δ() also takes a scheduling function $S_p^{t-1}()$ which restricts Δ() to operate on a subset of the model A
- the model parameters are not independent
- each parallel worker contributes equally

PETUUM

System design

Petuum Goal: **allow users to easily implement data-parallel and model-parallel ML algorithms!**

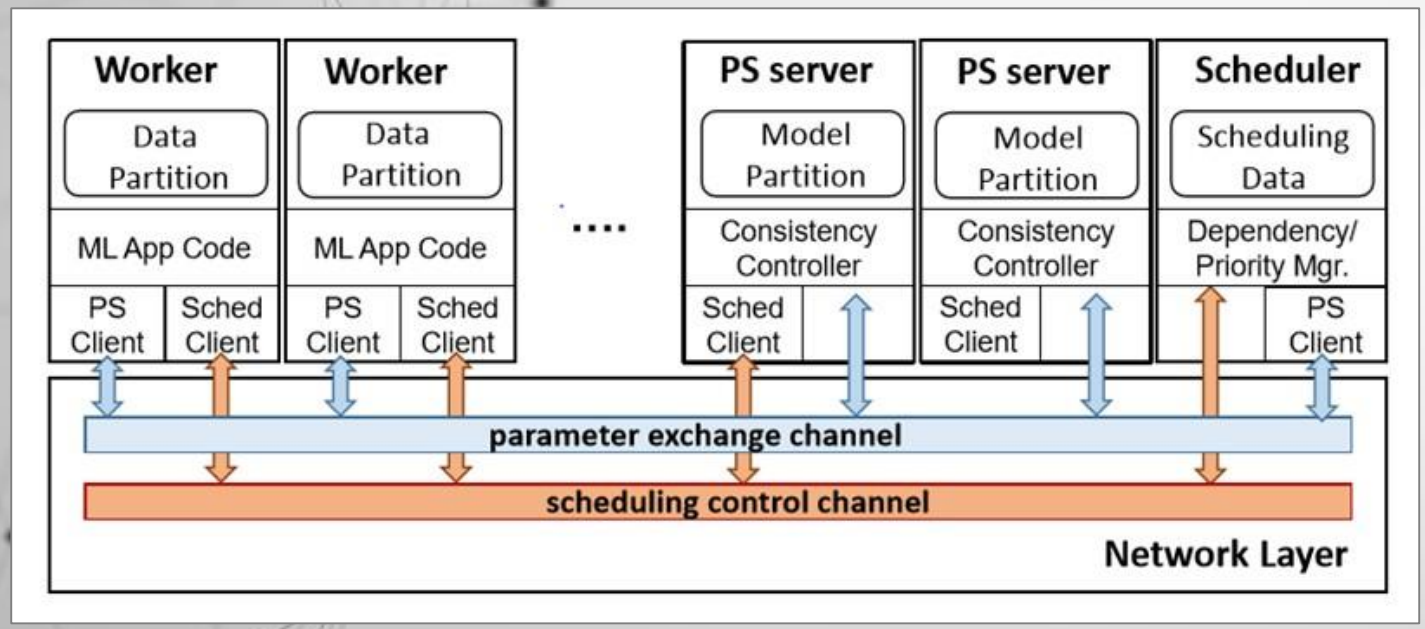


Fig. 5: Petuum scheduler, workers, parameter servers.

Parameter Server (PS):

- enables data-parallelism, by providing users with global read/write access to model parameters
- three functions: `PS.get()`, `PS.inc()`, `PS.put()`

Scheduler:

- enables model-parallelism, by allowing users to control which model parameters are updated by worker machines
- scheduling function `schedule()` - outputs a set of parameters for each worker

Workers:

- receives parameters to be updated from `schedule()`, and then runs parallel update functions `push()`

System design

BÖSEN

- **parameter server for data-parallel Big Data AI & ML**
- uses a consistency model, which allows asynchronous-like performance and bulk synchronous execution, yet does not sacrifice ML algorithm correctness

STRADS

- **scheduler for model-parallel High Speed AI & ML**
- performs fine-grained scheduling of ML update operations
- prioritizing computation on the parts of the ML program that need it most
- avoiding unsafe parallel operations that could hurt performance

POSEIDON

- **distributed GPU deep learning framework based on Caffe and supported by Bösen**
- enjoys speedups from the communication and bandwidth management features of Bösen

PETUUM

YARN compatibility

Problem:

- many industrial and academic clusters run Hadoop → MapReduce + YARN + HDFS
- however, programs that are written for stand-alone clusters are not compatible with YARN/HDFS, and vice versa, applications written for YARN/HDFS are not compatible with stand alone clusters.

Solution:

- providing common libraries that work on either Hadoop or non-Hadoop clusters
  - YARN launcher that will deploy any Petuum application (including user-written ones) onto a Hadoop cluster
  - a data access library for HDFS access, which allows users to write generic file stream code that works on both HDFS files the local filesystem.

PETUUM

Performance

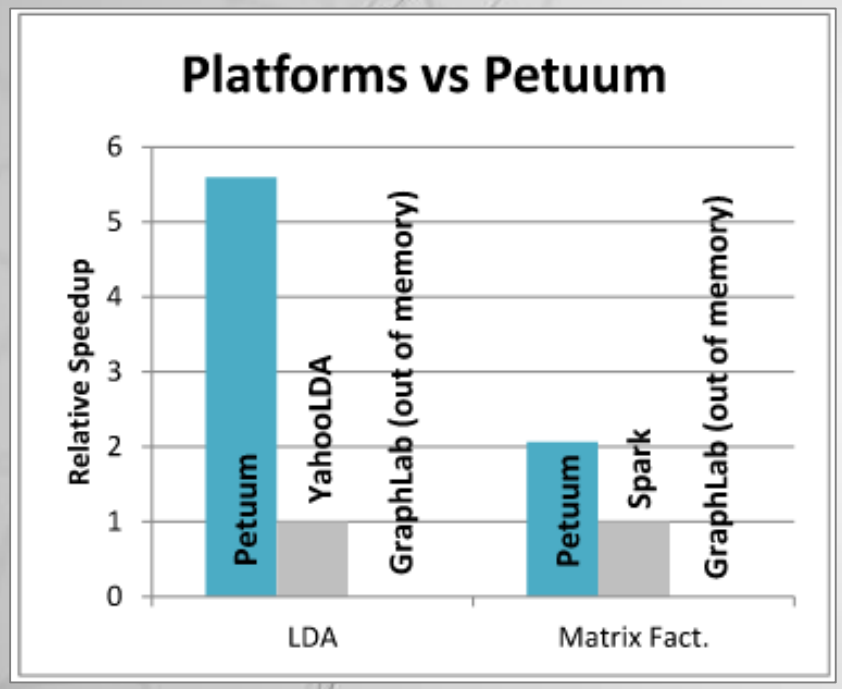


Fig. 6: Petuum relative speedup vs popular platforms (larger is better). Across ML programs, Petuum is at least 2-10 times faster than popular implementations.

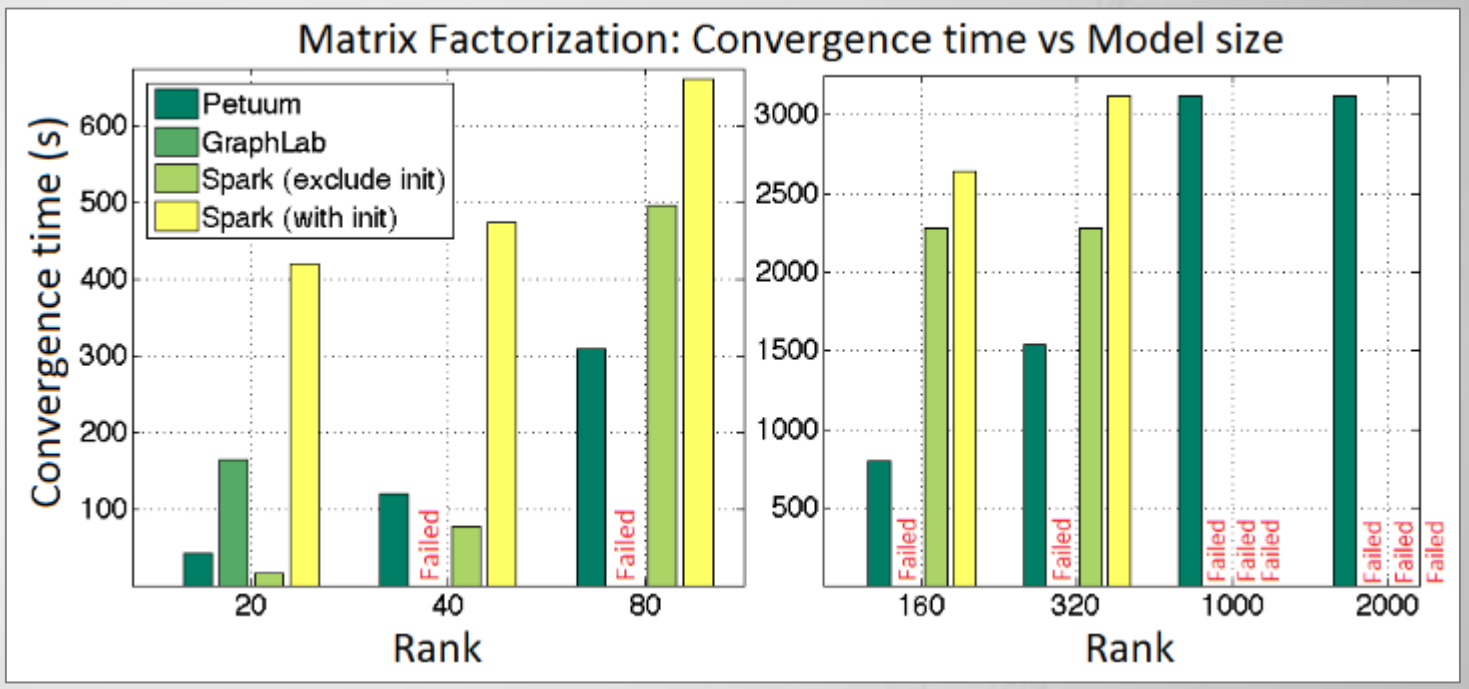


Fig. 7: Matrix Factorization convergence time: Petuum vs GraphLab vs Spark. Petuum is fastest and the most memory-efficient, and is the only platform that could handle Big MF models with rank K ≥ 1000 on the given hardware budget.

PETUUM

Example

```
// Petuum Program Structure

schedule() {
  // This is the (optional) scheduling function
  // It is executed on the scheduler machines
  A_local = PS.get(A)  // Parameter server read
  PS.inc(A,change)  // Can write to PS here if needed
  // Choose variables for push() and return
  svars = my_scheduling(DATA,A_local)
  return svars
}

push(p = worker_id(), svars = schedule()) {
  // This is the parallel update function
  // It is executed on each of P worker machines
  A_local = PS.get(A)  // Parameter server read
  // Perform computation and send return values to pull()
  // Or just write directly to PS
  change1 = my_update1(DATA,p,A_local)
  change2 = my_update2(DATA,p,A_local)
  PS.inc(A,change1)  // Parameter server increment
  return change2
}

pull(svars = schedule(), updates = (push(1), ..., push(P)) ) {
  // This is the (optional) aggregation function
  // It is executed on the scheduler machines
  A_local = PS.get(A)  // Parameter server read
  // Aggregate updates from push(1..P) and write to PS
  my_aggregate(A_local,updates)
  PS.put(A,change)  // Parameter server overwrite
}
```

Fig. 8: Petuum Program Structure

```
// Data-Parallel Distance Metric Learning

schedule() { // Empty, do nothing }

push() {
  L_local = PS.get(L) // Bounded-async read from param server
  change = 0
  for c=1..C    // Minibatch size C
    (x,y) = draw_similar_pair(DATA)
    (a,b) = draw_dissimilar_pair(DATA)
    change += DeltaL(L_local,x,y,a,b)  // SGD from Eq 7
  PS.inc(L,change/C)  // Add gradient to param server
}

pull() { // Empty, do nothing }
```

Fig. 9: Petuum DML data-parallel pseudocode

PETUUM

References

- [PETUUM A New Platform for Distributed Machine Learning on Big Data Eric] P. Xing, Qirong Ho Wei Dai, Jin Kyu Kim Jinliang Wei, Seunghak Lee Xun Zheng, Pengtao Xie Abhimanu Kumar, Yaoliang Yu. IEEE Transactions on Big Data, June 2015
- https://github.com/petuum
- http://www.simplilearn.com/how-facebook-is-using-big-data-article

Thank You !

PETUUM