FlexPS: A flexible and scalable Parameter Server for Distributed ML

ESTR4999 Graduation Thesis II

Name: Zhanhao Liu
Supervisor: Prof. James Cheng

https://github.com/Yuzhen11/flexps

4 Needs for ML & AI Tech

Resource **Efficient:** *Use less CPU, GPU, memory, networks …*

Scalable: *Linear performance increase with more computation resource*

Correct: *Can we trust the result?*

General ML/AI Platform: *Deep learning is only 10%*

dmlc mxnet TensorFlow torch Microsoft CNTK

Efficient & correct

?

Efficient, Scalable, Correct & General?

hadoop APACHE Spark

Correct & general

Big Data

facebook

1B+ Users
30+ Petabytes

WIKIPEDIA
The Free Encyclopedia

32M+ pages

YouTube

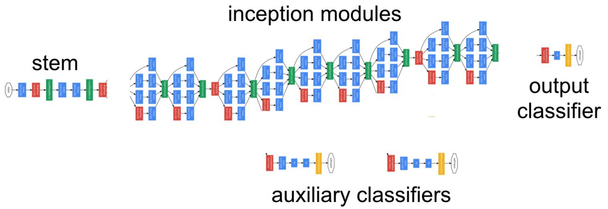
100h+ video uploaded
every minute

twitter

645 millions users
500 millions tweets / day

Big Model

Large Model is better for Big Data



Google Brain deep learning for images:

1 ~ 10 billion model parameters

Movie Ratings	Zora	Sophie	Jordan	Ernie	Christie
Harold and Kumar Escape..	8	4	?	?	4
Ted	?	?	8	10	4
Straight Outta Compton	8	10	?	?	6

Netflix collaborative filtering for video recommendation:

1 ~ 10 billion model parameters

Topic Models for text analysis:

Up to 1 Trillion model parameters

Iterative Convergent Algorithm

Data: *D*

Model *L* (ie. a fitness function such as likelihood)

Algorithm: update the model's parameters *A* iteratively until it converges

$$A^t = F(A^{t-1}, \Delta_L(A^{t-1}, D))$$

Algorithms: Stochastic Gradient Descent (SGD), Stochastic Average Gradient (SAG), stochastic variance reduced gradient (SVRG).

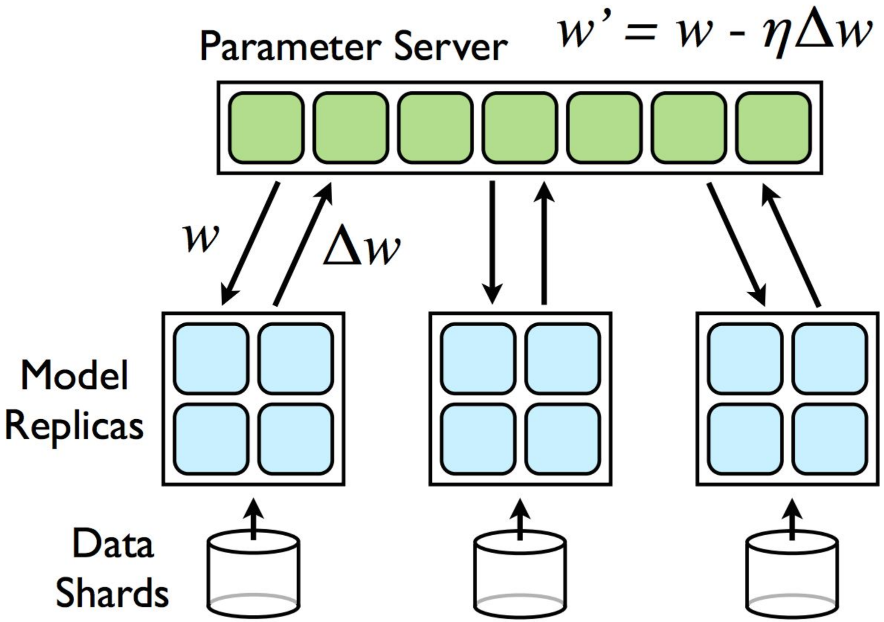
Models: Logistic Regression, Support Vector Machine, Kmeans, Neural Network

Parameter Server

separate the working units into *workers* and *servers*

- Parallel workers update models stored distributedly in servers

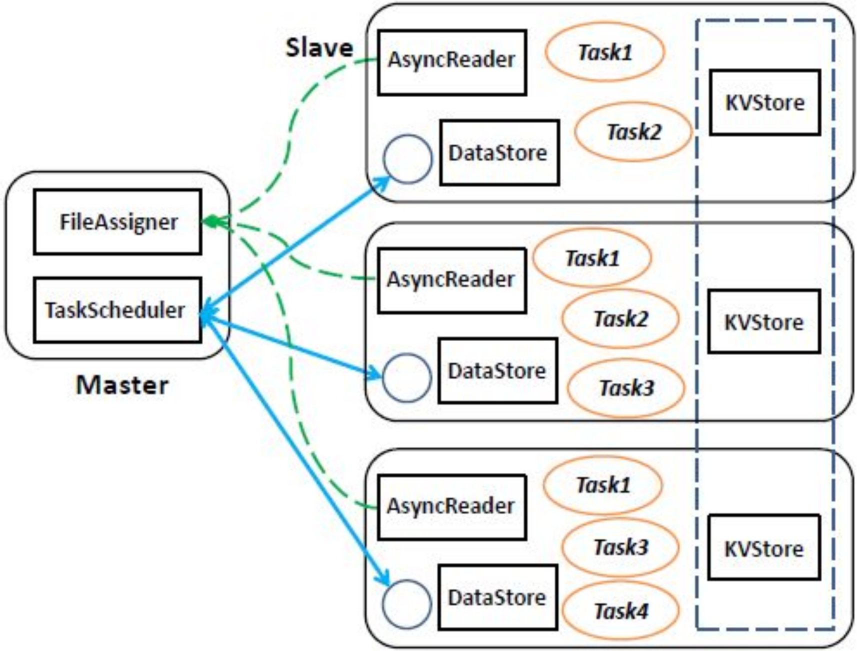
- Easy-to-use key/value store interface



FlexPS: Architecture

FlexPS is organized in a master/slave architecture

- Master
  - TaskScheduler
  - FileAssigner
- Slave
  - KVStore
  - DataStore
  - AsynReader
  - Tasks

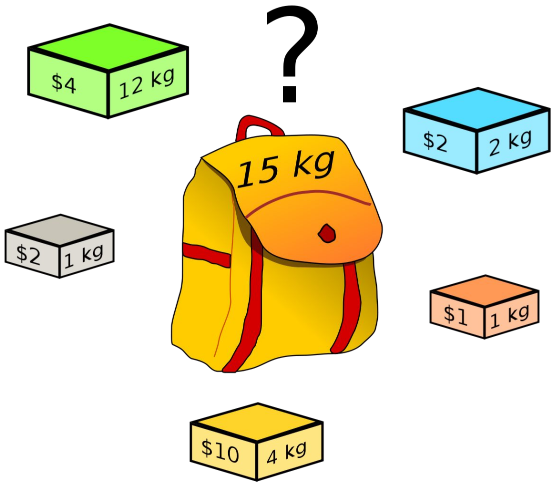


Architecture: TaskScheduler

TaskScheduler decides which task to run & where it runs.

3 scheduling algorithms:

- Sequential scheduling
- Greedy scheduling
- Prioritized scheduling

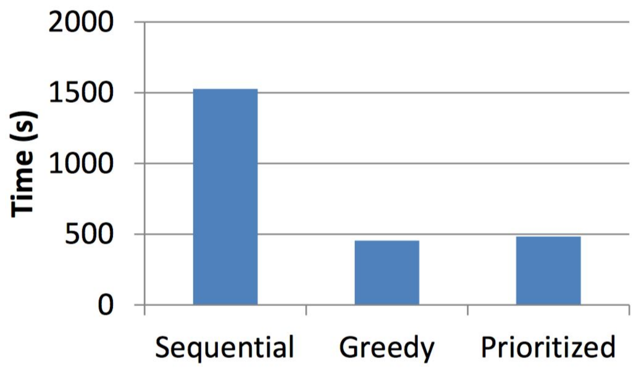


Optimal scheduling algorithm?

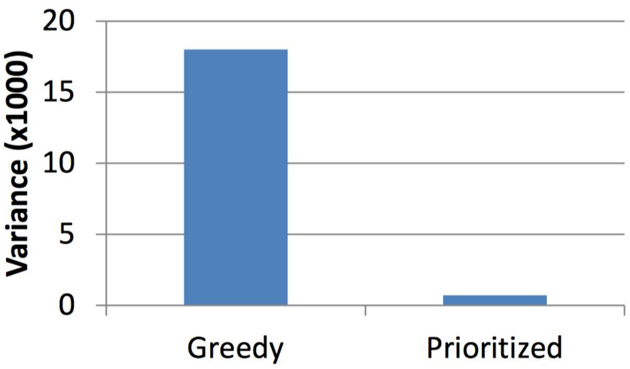
Multiple Knapsack Problem → NP Hard

Evaluation: Task Scheduling

- **6** tasks with different workloads, **300** threads
- Greedy and Prioritized significantly outperform the Sequential due to task parallelism
- Prioritized: prevent starvation & high throughput



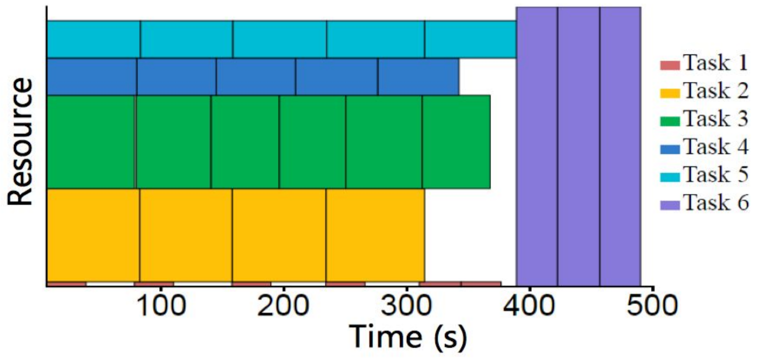
(a) Total running time



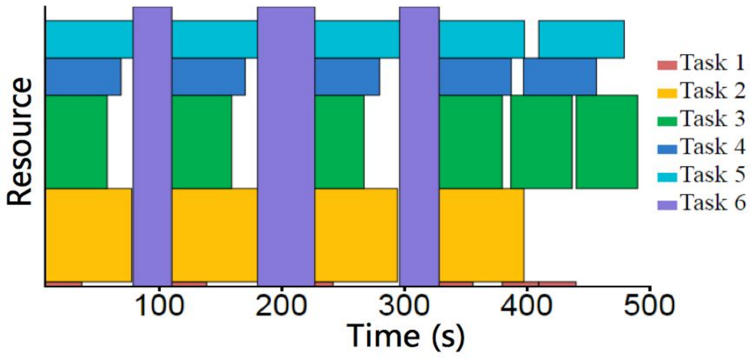
(b) Starting time variance

Evaluation: Task Scheduling

- **6** tasks with different workloads, **300** threads
- Resource → # of threads
- Both algorithms achieve high resource utilization at most time
- Prioritized scheduling eliminates starvation (Task 6)



(a) Greedy



(b) Prioritized

Architecture: KV-Store

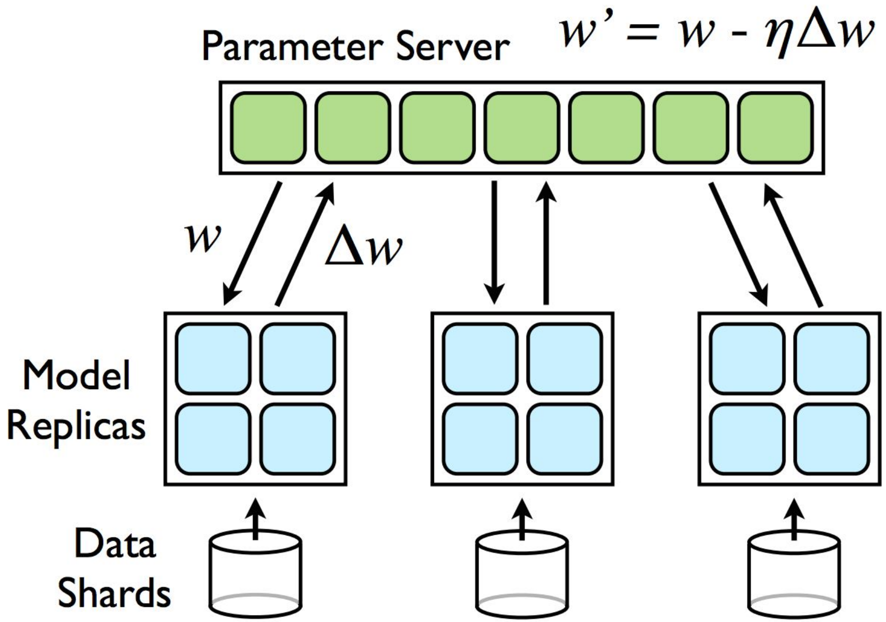
Programming Model:

Get(keys)

Add(keys, vals)

GetChunk(keys)

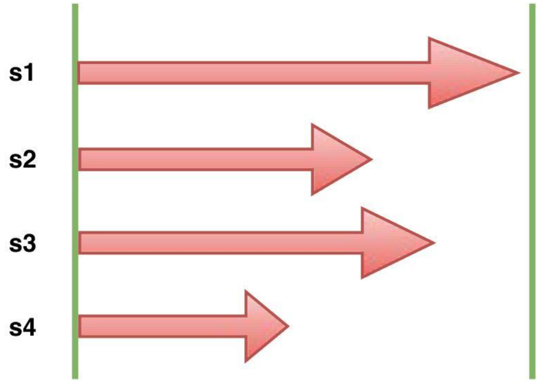
AddChunk(keys, vals)



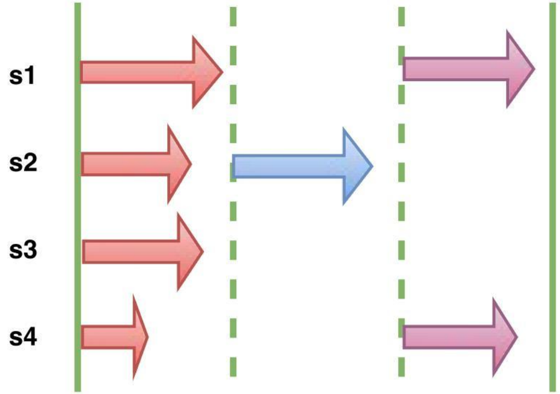
Key Design: Multi-stage ML

Breaks down a machine learning task into different stages

A *stage* runs a sub-task on a specific set of computing resources characterized by the number of slaves and the location of these slaves



Traditional PS (one stage)

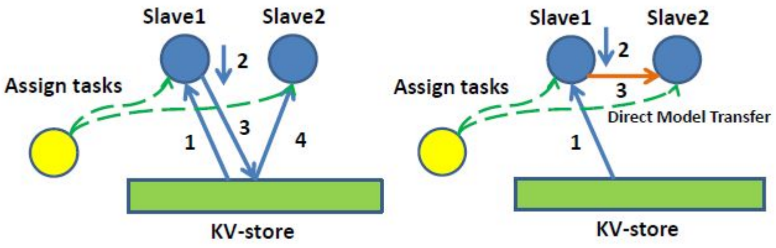


FlexPS (multi-stage)

Optimization: Direct Model Transfer

Normal Procedure:

1. Load the model from KV-Store
2. Perform training locally
3. Dump the model to KV-Store
4. Repeat above steps in subsequent stages



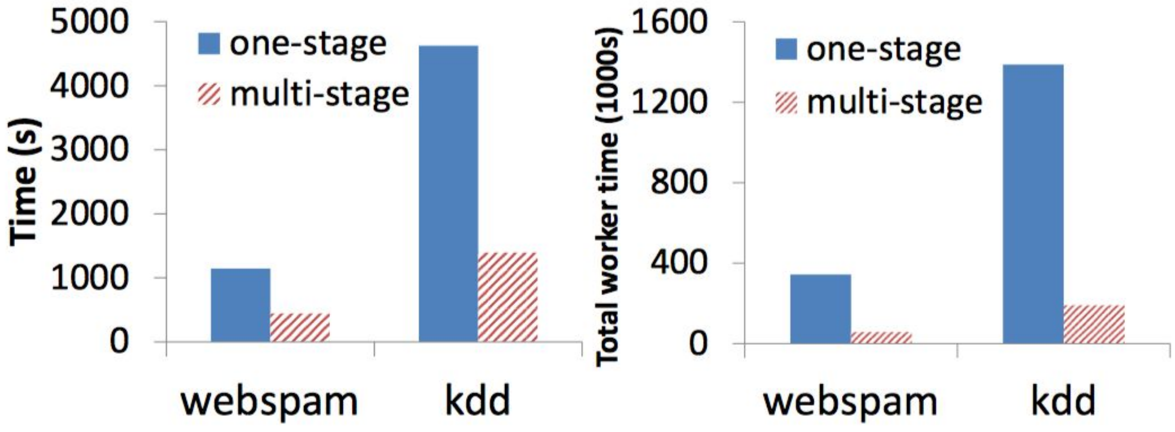
(a) Changing stages w/o DMT (b) Changing stages w/ DMT

Direct Model Transfer:

- Bypass KV-Store
- Boost the performance of changing stages by 23%

Evaluation: LR Performance on SVRG

- Total worker time: the total amount of (worker × time) in all stages (reflect total CPU hours)
- 3.5x on elapsed time, 8x on worker time



(c) Total elapsed time (d) Total worker time

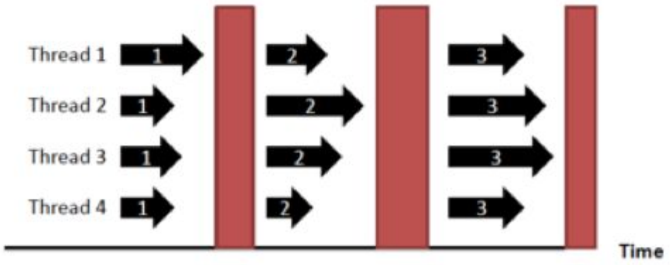
Background: Consistency models

Bulk Synchronous Parallel(BSP)

Synchronization Barrier (Parameters read/updated here)

(a) Machines perform unequally

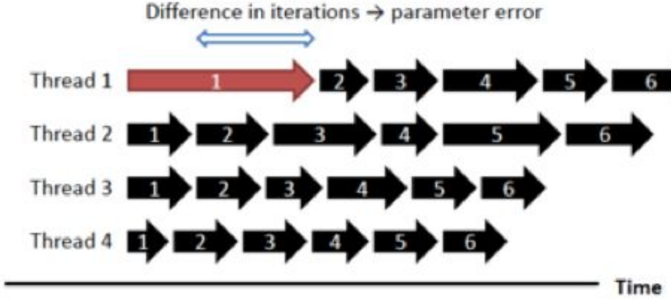
(b) Algorithmic workload imbalanced



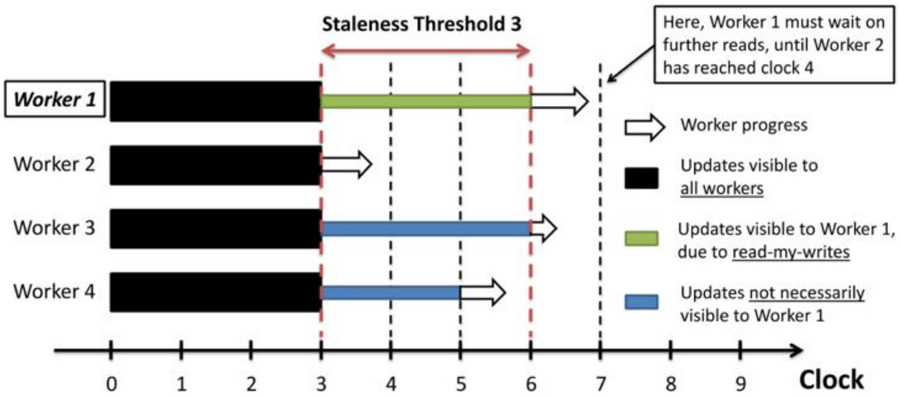
Asynchronous

Parameters read/updated at any time

Asynchronous is fast but has weak convergence guarantees



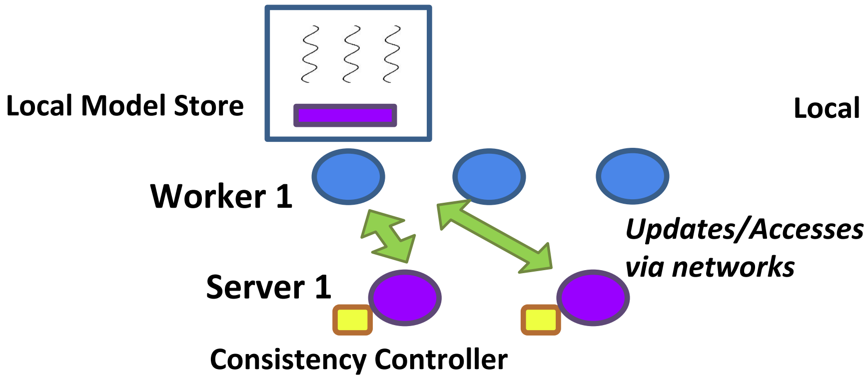
SSP: Bounded Staleness and Clocks

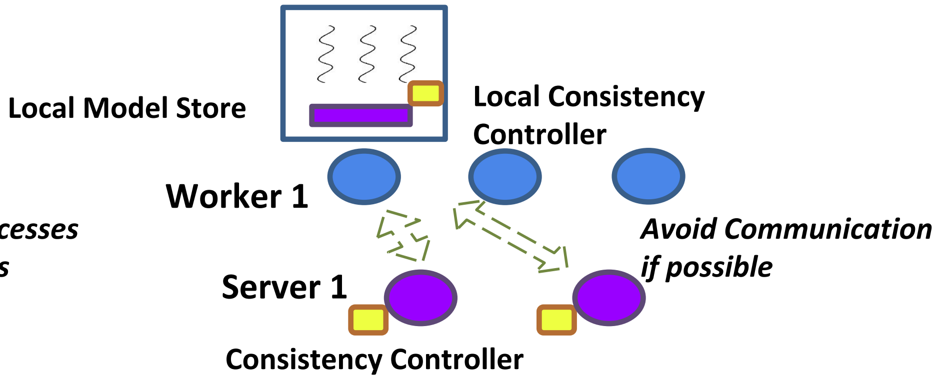


Ho, Q. (NIPS 2013). More E ective Distributed ML via a Stale Synchronous Parallel Parameter Server.

Key Design: Local Consistency Control

Pop up consistency controller from server side

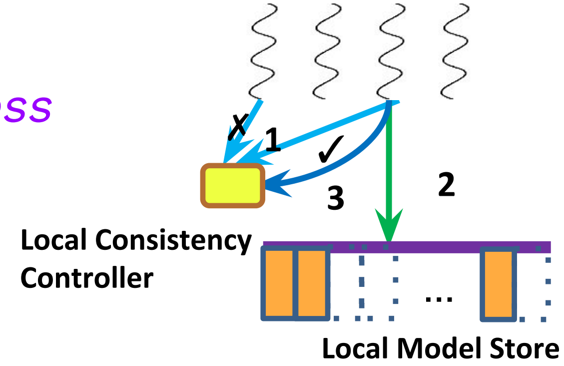




Key Design: Local Consistency Control

Model access procedure:

1. working thread checks with the local consistency controller
2. If the consistency requirement is satisfied, this thread is granted to access the model
3. When slowest thread finish, update *min_progress* & notify blocked threads



All updates & consistency controls are performed locally!

Key Design: Local Consistency Control

Concurrency control:

- Multiple threads may access the shared local model simultaneously
- Sort & divide the local model into chunks
- Attach a lock (mutex) to each chunk
- Tradeoff: Lock every single parameter vs Lock a chunk
- Experiments: Less contention, fast model access (1-2% of total time)

FlexPS: Optimizations

Load Balancing:

A global *FileAssigner* to keep the block information of all files in HDFS

Assigns blocks to the worker threads according to *data locality*

Data loading on-the-fly:

AsyncReader module with the classical producer-consumer paradigm

reader threads load the data from HDFS and store them to a pre-allocated buffer

worker threads consume the data in the buffer and use the data to train the model

Comparison: LR with SVRG

FlexPS-Opt: offline search for optimal stage config

FlexPS-Auto: runtime search for stage config

FlexPS-: disable the flexible parallelism control of FlexPS

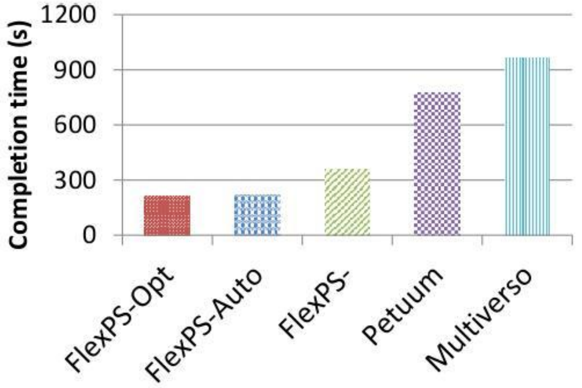
Dataset: webspam, kdd

Batch size of the stochastic step:

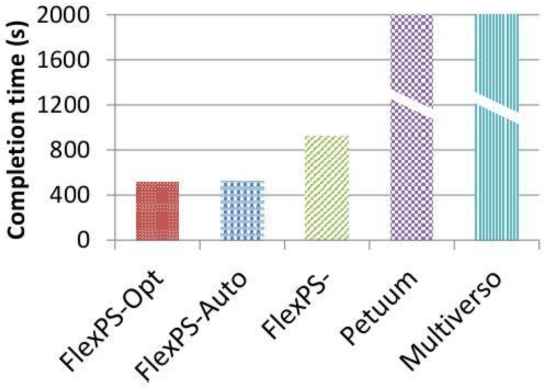
0.1% for webspam and 0.001% for kdd

Number of training epochs: 10

2-5 times speedup in task completion time!



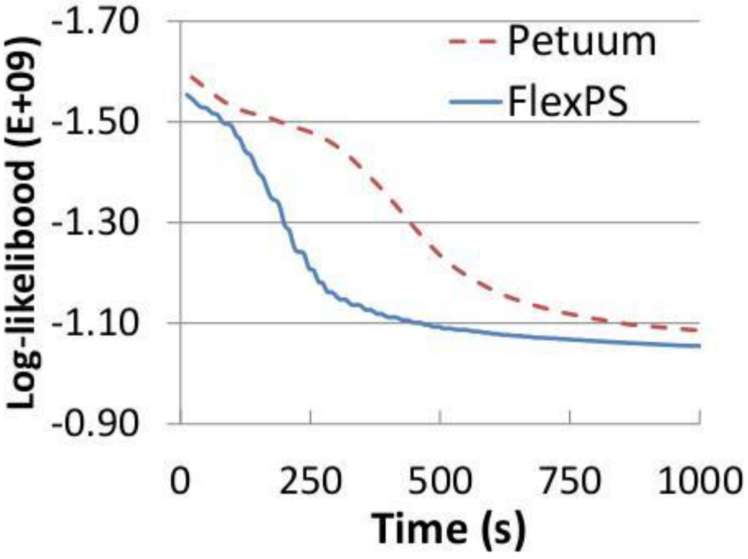
(a) webspam



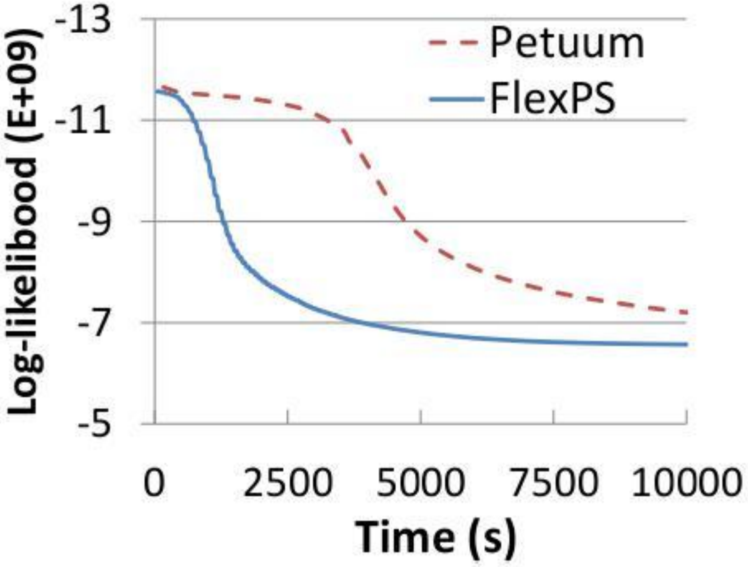
(b) kdd

http://www.cc.gatech.edu/projects/doi/WebbSpamCorpus.html
www.kddcup2012.org/c/kddcup2012-track2/data

Comparison: Convergence speed on LDA



(a) NYTimes



(b) Pubmed

Conclusion

We proposed FlexPS, which provides a more *flexible* and *scalable* PS framework for distributed ML with innovative designs:

- TaskScheduler: flexible task scheduling (sequential, greedy, prioritized)
- Multi-stage design: make the most of the computing resources
  - Direct model transfer: Bypass the KV-Store
- Local consistency control: Avoid network traffic
  - Concurrency control: Lock single parameter vs Lock a chunk
- Load Balancing: exploit the data locality
- Data load on-the-fly: asynchronous data loading
- ...

Q&A

Example: Logistic Regression

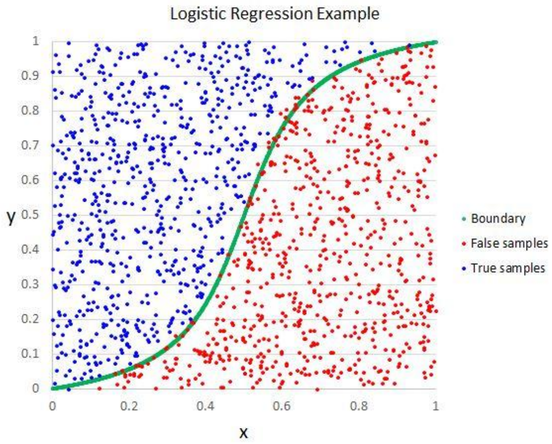
y = *w*.*X* + b

y: Discrete class [0, 1] prediction

X: data set (a matrix)

Cost: Correct/Wrong prediction from actual

Goal: Find scalars w, b



$$P(y=1|x) = h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$
$$P(y=0|x) = 1 - P(y=1|x) = 1 - h_\theta(x).$$

Iterative Convergent Algorithm, often solved by SGD

Example: Stochastic Variance Reduced Gradient

Procedure SVRG

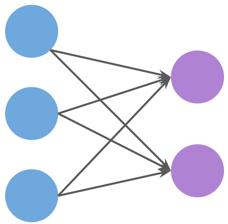
Parameters update frequency m and learning rate η
Initialize $\tilde{w}_0$
Iterate: for $s = 1, 2, \ldots$
  $\tilde{w} = \tilde{w}_{s-1}$
  $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \nabla\psi_i(\tilde{w})$
  $w_0 = \tilde{w}$
  Iterate: for $t = 1, 2, \ldots, m$
    Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
      $w_t = w_{t-1} - \eta(\nabla\psi_{i_t}(w_{t-1}) - \nabla\psi_{i_t}(\tilde{w}) + \tilde{\mu})$
  end
  option I: set $\tilde{w}_s = w_m$
  option II: set $\tilde{w}_s = w_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
end

Example: Multi-stage LR with SVRG

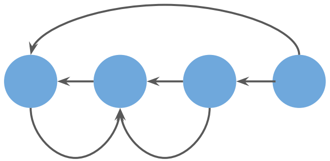
```
/* Step 1: define the stages */
auto fgd_lambda = [](Info info) {
// Get model parameter w from the KV-store
// Calculate gradient
// Update full gradient u in the KV-store
};
auto sgd_lambda = [](Info info) {
// Get w and u from the KV-store
// Calculate gradient
// Perform variance-reduced update
// Update w to the KV-store
};
/* Step 2: set the parallelism degree */
MultiStageTask task;
task.SetStages({{fgd_lambda, 100},{sgd_lambda, 10}});
/* Step 3: submit the task */
engine.SubmitAndWait(task);
```

Machine Learning Systems

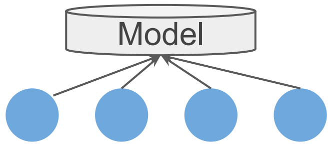
Iterative MapReduce



Data Flow Graph



Parameter Server



hadoop

Spark MLlib

TensorFlow

PETUUM

dmlc mxnet

Why Parameter Server?

Support for big model

The driver can't hold big model due to limited memory size

Flexible Consistency Control

Allows asynchronous operation

Spark and Hadoop are Bulk Synchronous Parallel

Better network utilization, and lets you scale your models

Existing PS Systems

ps-lite:

Underlying communication module of *MXNet*

Flexible consistency models, Fault Tolerance

Petuum:

Developed by SAILING Lab from CMU, now a start-up company

Flexible consistency models (SSP), Model parallelism

Multiverso:

Core module of the Microsoft Distributed Machine Learning Toolkit (DMTK)

Abstract Communication APIs, Supports for deep learning systems (torch, theano)

Limitations in Existing PS Systems

Flexibility in Parallelism Control

Machine learning algorithms may have varying workloads in different training stages

Stochastic algorithm: smaller mini-batch sizes in earlier stages for fast convergence and larger mini-batches later to avoid oscillation

variance reduction algorithms: repeats the following two phases until convergence:

1. Compute the full gradient using all data records
2. Update the parameters in a stochastic manner

Existing PS Systems only allow a *fixed* number of worker threads (i.e., fixed parallelism) throughout the whole training process

Limitations in Existing PS Systems

Optimization for Sparse Data

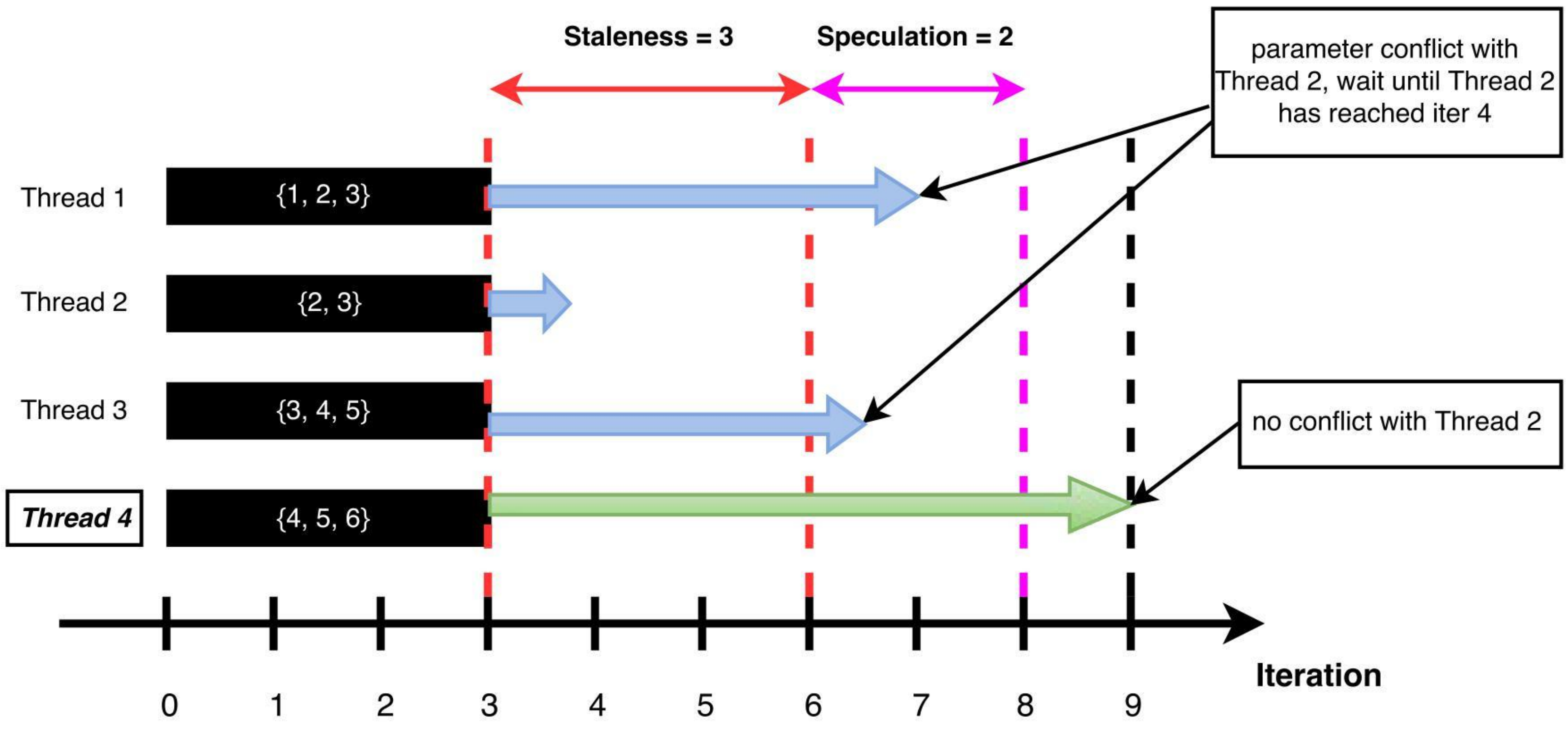
In most cases of the high dimension dataset, the data is actually sparse (data missing).

Only store the non-zero entries as key-value pairs to save memory.

Special optimization for sparse machine learning problem with high-dimensional parameters needs to be proposed for lower computational/communication cost.

Existing PS Systems typical support communication models like BSP/SSP/ASP, which make no difference between sparse and dense data.

FlexPS: SparseSSP



FlexPS: SparseSSP

Further optimize SSP for sparse data

Allow worker move to next iteration if the set of parameter that it is manipulating does not conflict with the slowest worker’s

Max clock difference between workers:

Staleness + Speculation

Algorithm 1 SparseSSP Algorithm to check if worker i can move on to next iteration

Require: w_i: worker i c_i: the clock value w_i s: staleness sp: speculation P_i: the set of parameter that w_i is manipulating

initialize flag = True
for every worker w_j from all worker where $i \neq j$ **do**
    if $c_i - c_j \leq s$ **then**
        continue;
    else if $s < c_i - c_j <= s + sp$ **then**
        if $P_i \cap P_j = \emptyset$ **then**
            continue;
        end if
    end if
    flag = False;
    break;
end for
if flag = True **then**
    w_i can move on
else
    w_i waits until the slowest worker finish its current iteration
end if

How often will there be conflict?

Total number of parameter of dataset: n

Number of non-zero parameter in each training batch: m

Sparsity: m/n

Probobility of parameter conflict between two workers: $P(conflict) = 1 - \frac{\binom{n-m}{m}}{\binom{n}{m}}$

Dataset	kdd	avazu	criteo	url
# of features	5.5x10^7	10^6	10^6	3.2x10^6
# of samples	1.5x10^8	4.0x10^7	4.6x10^7	2.4x10^6
Sparsity	2.0x10^-7	1.5x10^-5	3.9x10^-5	3.1x10^-5
Pr of conflicts	2.2x10^-6	2.2x10^-4	1.5x10^-3	3.0x10^-3

Evaluation: SparseSSP

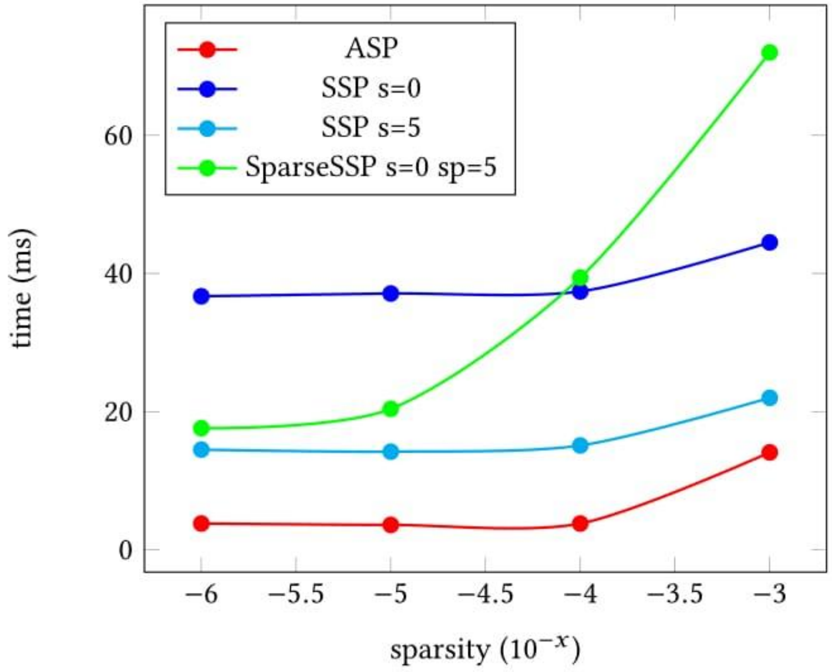


Figure 10: Comparison of different parallelism protocol in sparse data

Table 2: Experiment parameter

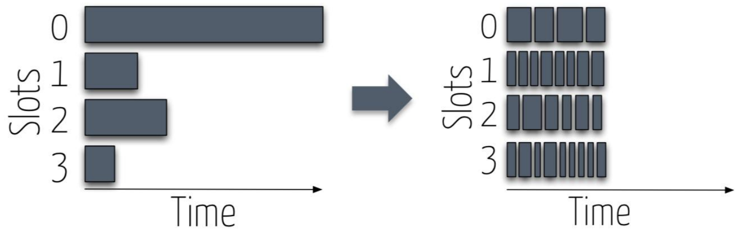
# of non-zero	10	100	1000	10000
Sparsity	10^{-6}	10^{-5}	10^{-4}	110^{-3}
Pr of conflicts	10^{-5}	10^{-4}	$9.5x10^{-2}$	0.9955

Future Work: Task based PS

Straggler Problem

The slowest worker in a cluster, cripples the scalibility of ditributed ML

Straggler mitigation via tiny tasks:



Task based Parameter Server:

High level & functional: (x;w) => delta

Abstract the threads, user only focus on defining ML task

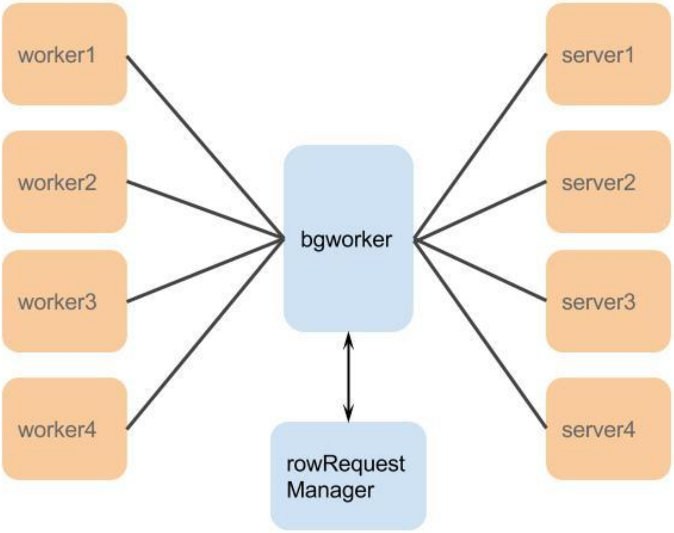
No need for SSP: tiny tasks --> load balancing

Process Cache

Currently, the worker need to
 fetch parameter from the server directly.

A cache could be created in the worker node
 to cache some temporary parameter.

It accelerate getChunk or AddChunk operations.



Load Balancing

Currently, load balancing is done during the loading data task, you could manually allocate data to different worker nodes.

In the future, load balancing could be done during the execution of task:

1. The worker threads will steal some data from other workers when it is blocked by consistency control.
2. The fast worker will help slow worker once the fast worker finish its own task.

Backup slides