Accelerating Machine Learning on Emerging Architectures

Big Simulation and Big Data Workshop

January 9, 2017
Indiana University

Judy Qiu

Associate Professor of Intelligent Systems Engineering
Indiana University

SALSA

Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work

Acknowledgements

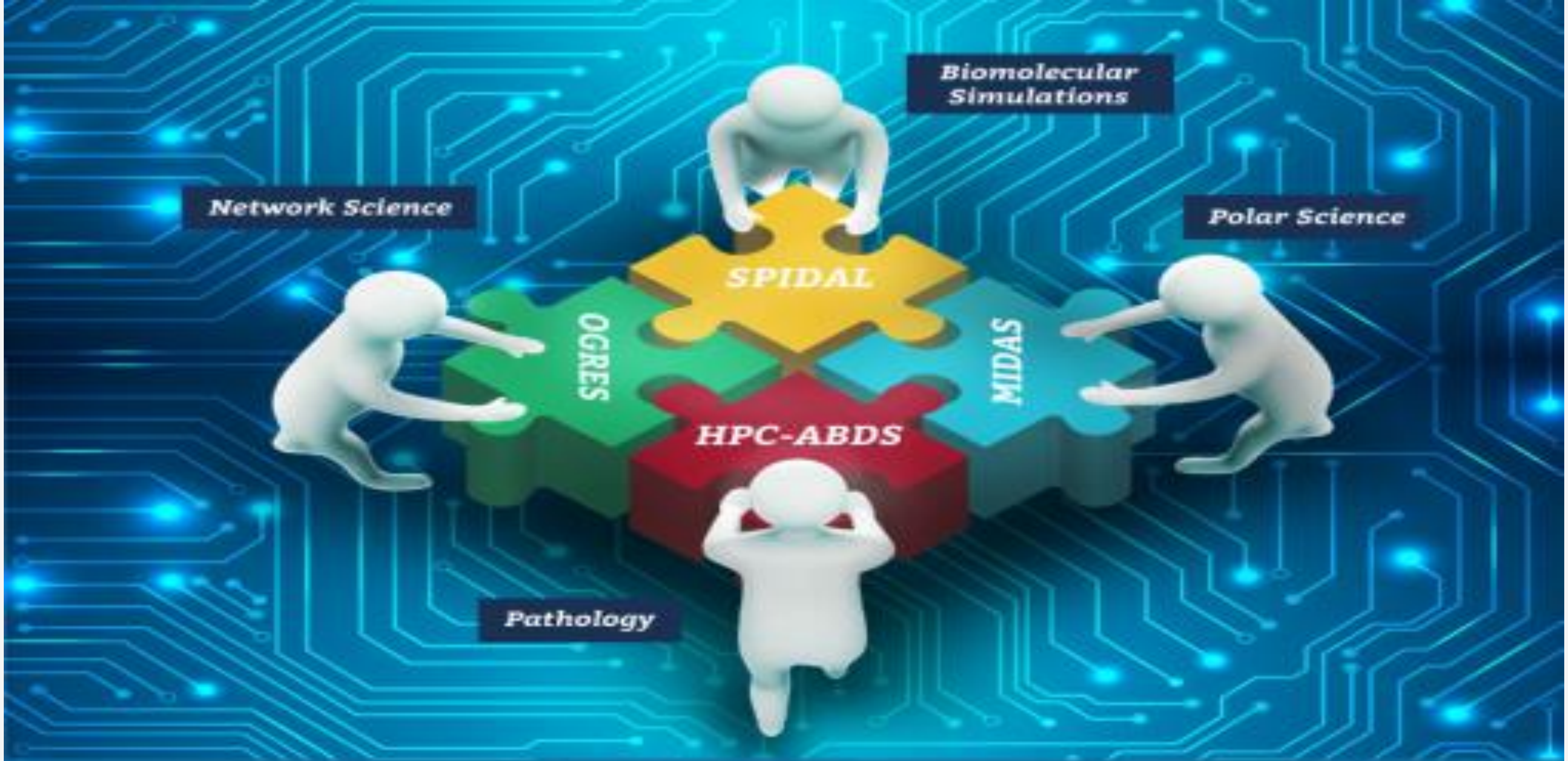
Bingjing Zhang | Yining Wang | Langshi Chen | Meng Li | Bo Peng | Yiming Zou

SALSA HPC Group
School of Informatics and Computing
Indiana University

Rutgers University
Virginia Tech
Kansas University
Arizona State University
State University of New York at Stony Brooke
University of Utah

Digital Science Center
Indiana University

Intel Parallel Computing Center
IPCC



DATANET: CIF21 DIBBS:
Middleware and High Performance Analytics Libraries for Scalable Data Science

Motivation

- Machine learning is widely used in data analytics
- Need for high performance
  - Big data & Big model
  - **”Select model and hyper parameter tuning"** step need to run the training algorithm for many times
- Key: optimize for efficiency
  - What is the 'kernel' of training?
  - Computation model

Recommendation Engine

- Show us products typically purchased together
- Curate books and music for us based on our preferences
- Have proven significant because they consistently boost sales as well as customer satisfaction

Fraud Detection

- Identify fraudulent activity
- Predict it before it has occurred saving financial services firms millions in lost revenue.
- Analysis of financial transactions, email, customer relationships and communications can help

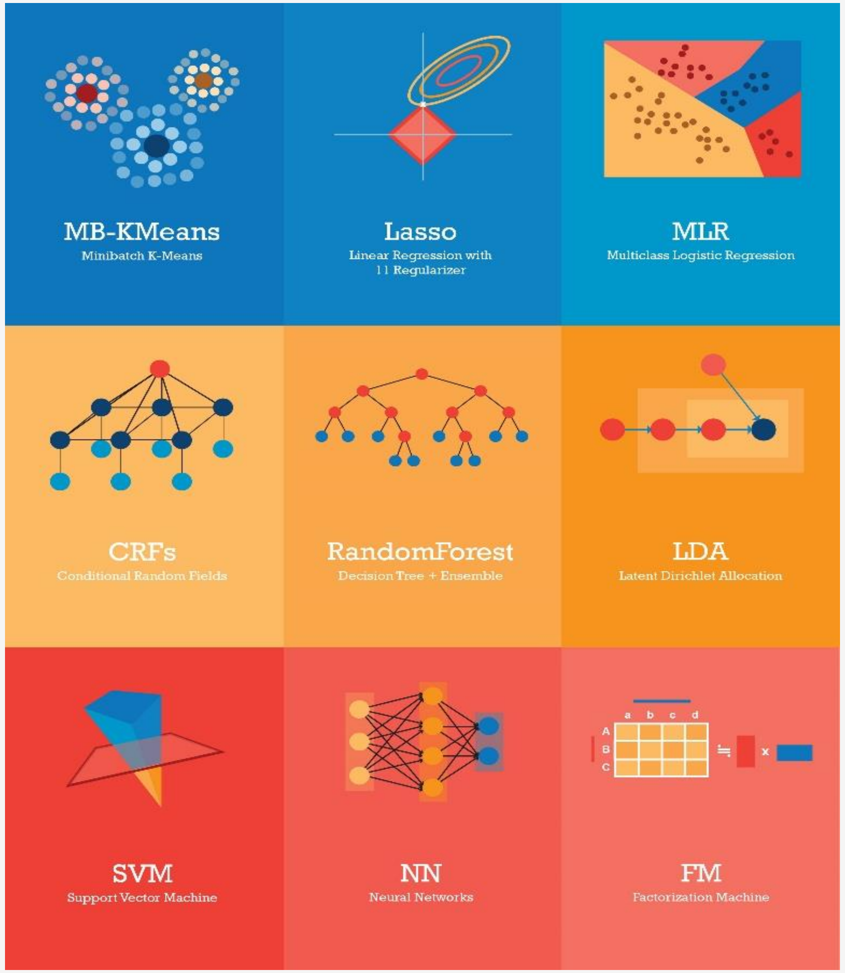
More Opportunities…

- Predicting customer “churn” – when a customer will leave a provider of a product or service in favor of another.
- Predicting presidential elections, whether a swing voter would be persuaded by campaign contact.
- Google has announced that it has used Deep Mind to reduce the energy used for cooling its datacenter by 40 per cent.
- Imagine…

The Process of Data Analytics

- **Define the Problem**
  - Binary or multiclass, classification or regression, evaluation metric, …
- **Dataset Preparation**
  - Data collection, data munging, cleaning, split, normalization, …
- **Feature Engineering**
  - Feature selection, dimension reduction, …
- **Select model and hyper paramenter tuning**
  - Random Forest, GBM, Logistic Regression, SVM, KNN, Ridge, Lasso, SVR, Matrix Factorization, Neural Networks, …
- **Output the best models with optimized hyper parameters**

Challenges from Machine Learning Algorithms



Machine Learning algorithms in various domains:

- Biomolecular Simulations
- Epidemiology
- Computer Vision

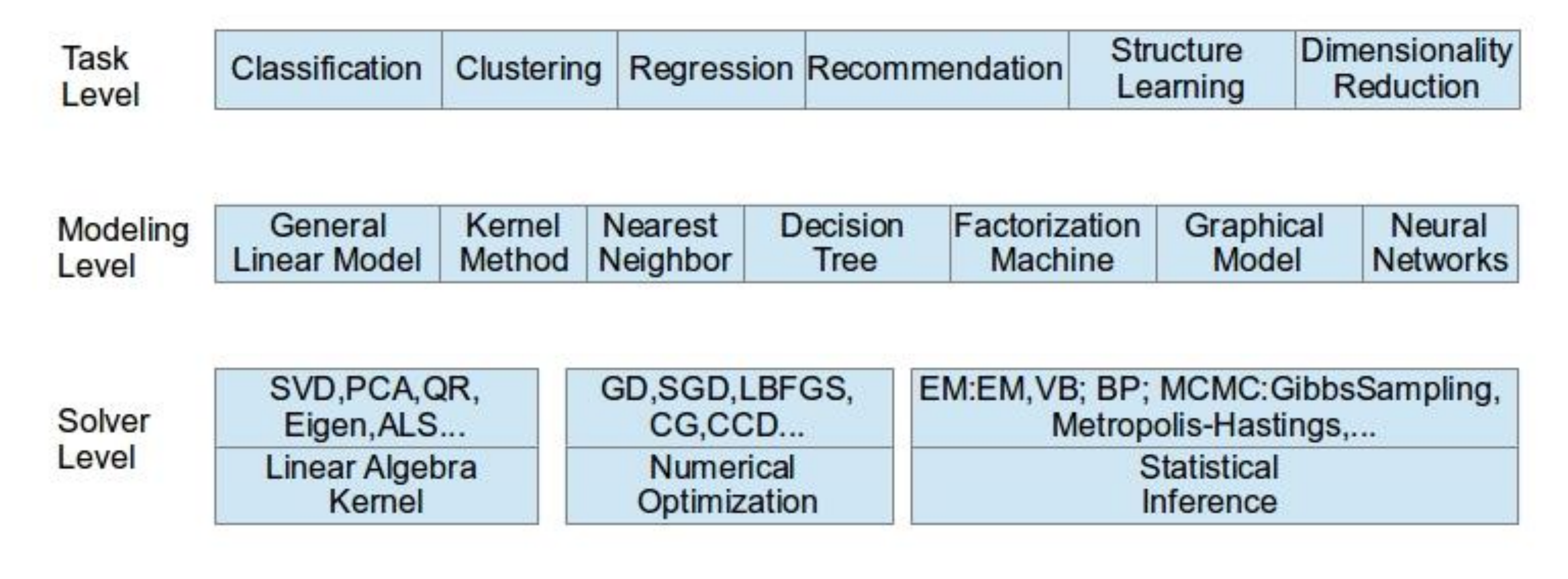
They have:

- Iterative computation workload
- High volume of training & model data

Traditional Hadoop/MapReduce solutions:

- Low Computation speed (lack of multi-threading)
- High data transfer overhead (disk based)

Taxonomy for ML Algorithms



Task Level

Classification | Clustering | Regression | Recommendation | Structure Learning | Dimensionality Reduction

Modeling Level

General Linear Model | Kernel Method | Nearest Neighbor | Decision Tree | Factorization Machine | Graphical Model | Neural Networks

Solver Level

SVD,PCA,QR, Eigen,ALS...
Linear Algebra Kernel

GD,SGD,LBFGS, CG,CCD...
Numerical Optimization

EM:EM,VB; BP; MCMC:GibbsSampling, Metropolis-Hastings,...
Statistical Inference

- Task level: describe functionality of the algorithm
- Modeling level: the form and structure of model
- Solver level: the computation pattern of training

Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work

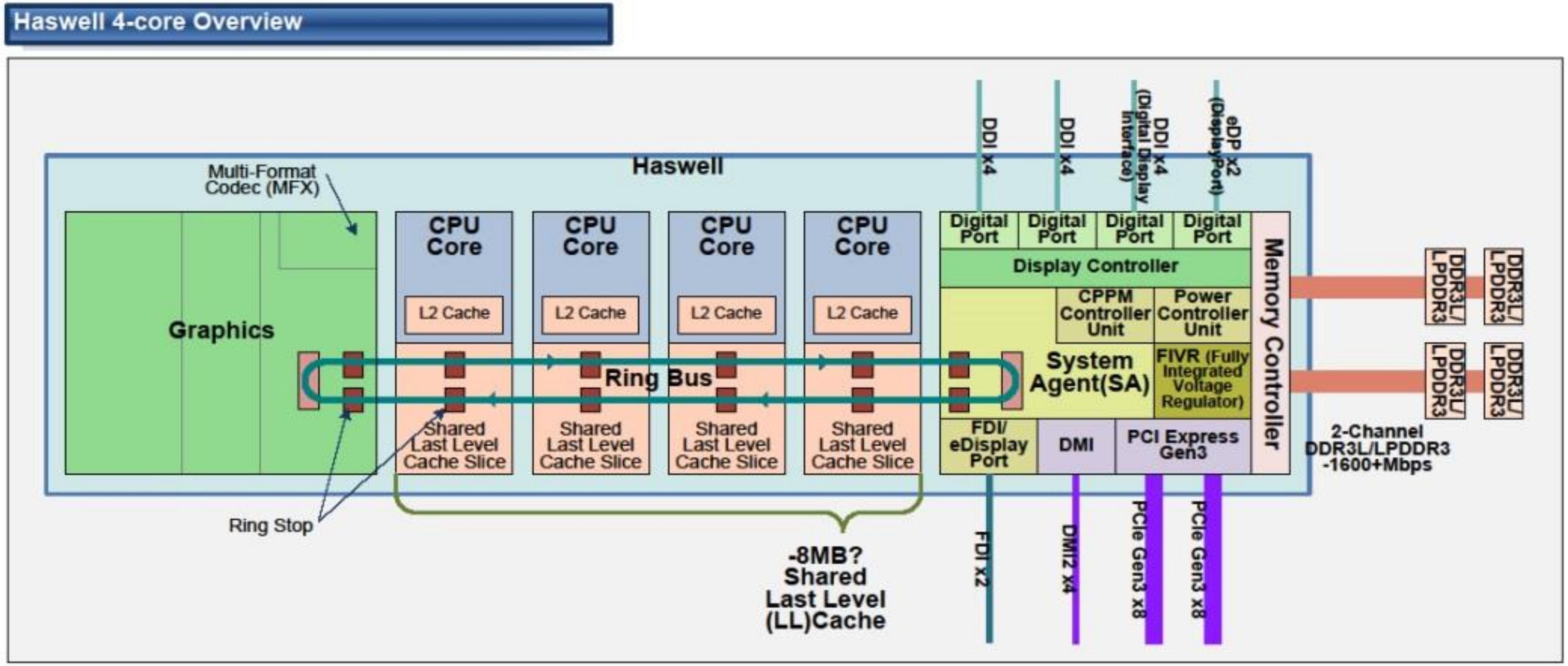
Emerging Many-core Platforms

Comparison of Many-core and Multi-core Architectures

- Much more number of cores
- Lower single core frequency
- Higher data throughput

How to explore computation and Bandwidth of KNL for Machine Learning applications ?

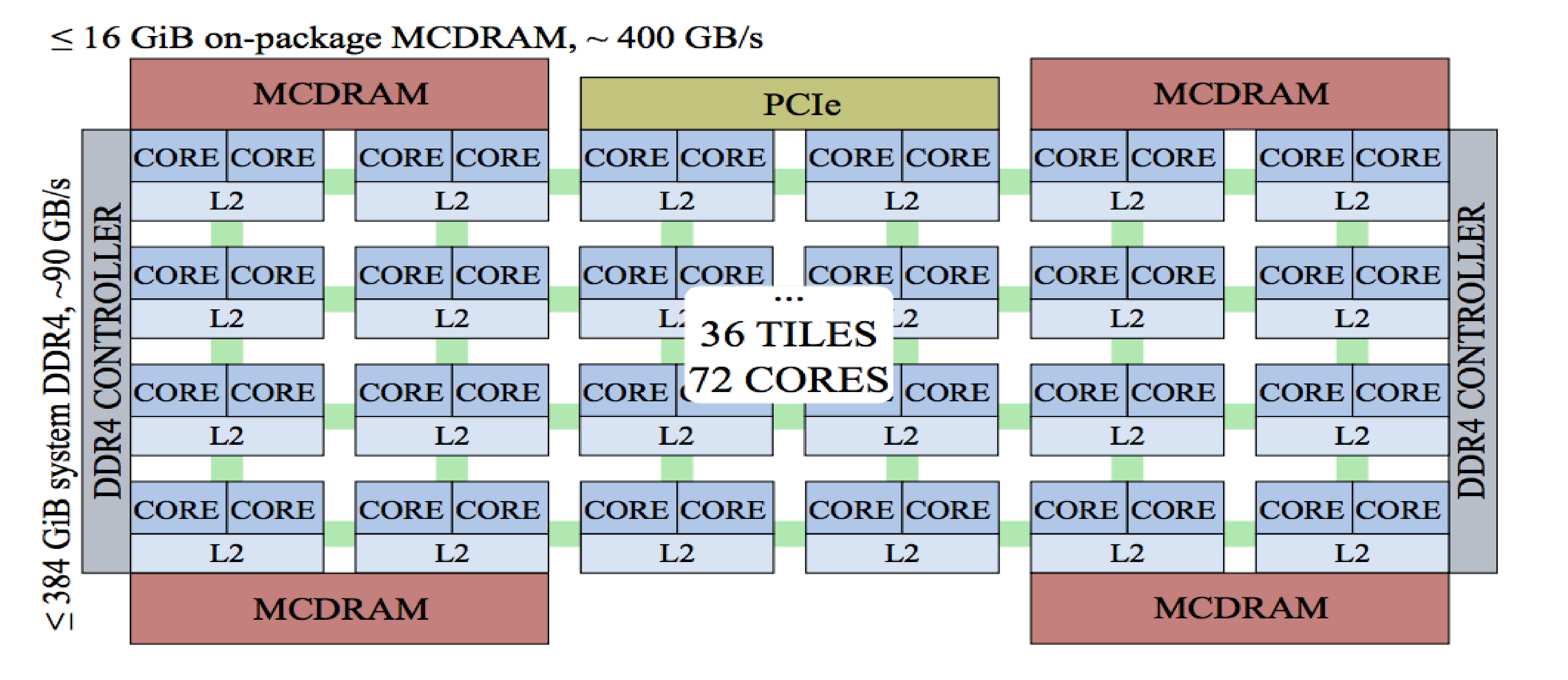
Intel Xeon/Haswell Architecture



Copyright (c) 2011 Hiroshige Goto All rights reserved.

- Much more number of cores
- Lower single core frequency
- Higher data throughput

Intel Xeon Phi (Knights Landing) Architecture

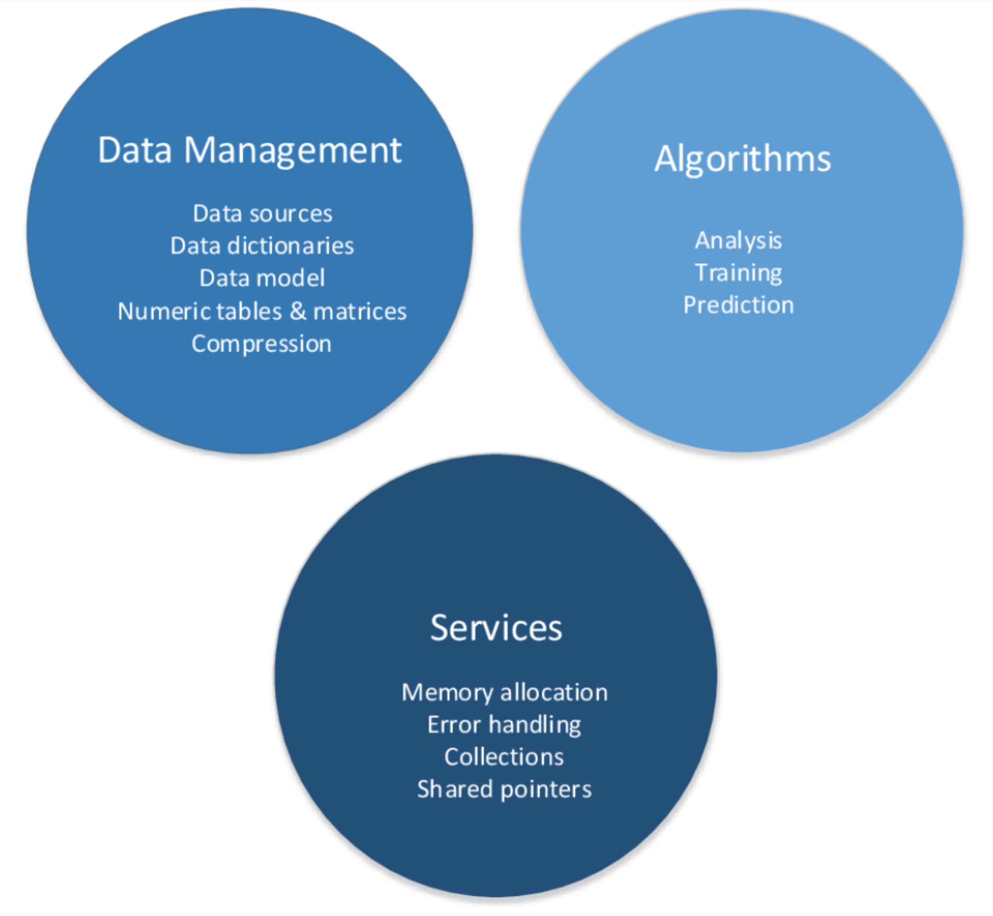


- Up to 72 cores, 288 threads connected in a 2D-mesh
- High bandwidth (> 400 GB/s) Memory (MCDRAM)

- Up to 144 AVX512 vectorization units (VPUs)
- 3 Tflops (DP) performance delivery
- Omni-path link among processors (~ 100 GB/s)

SALSA

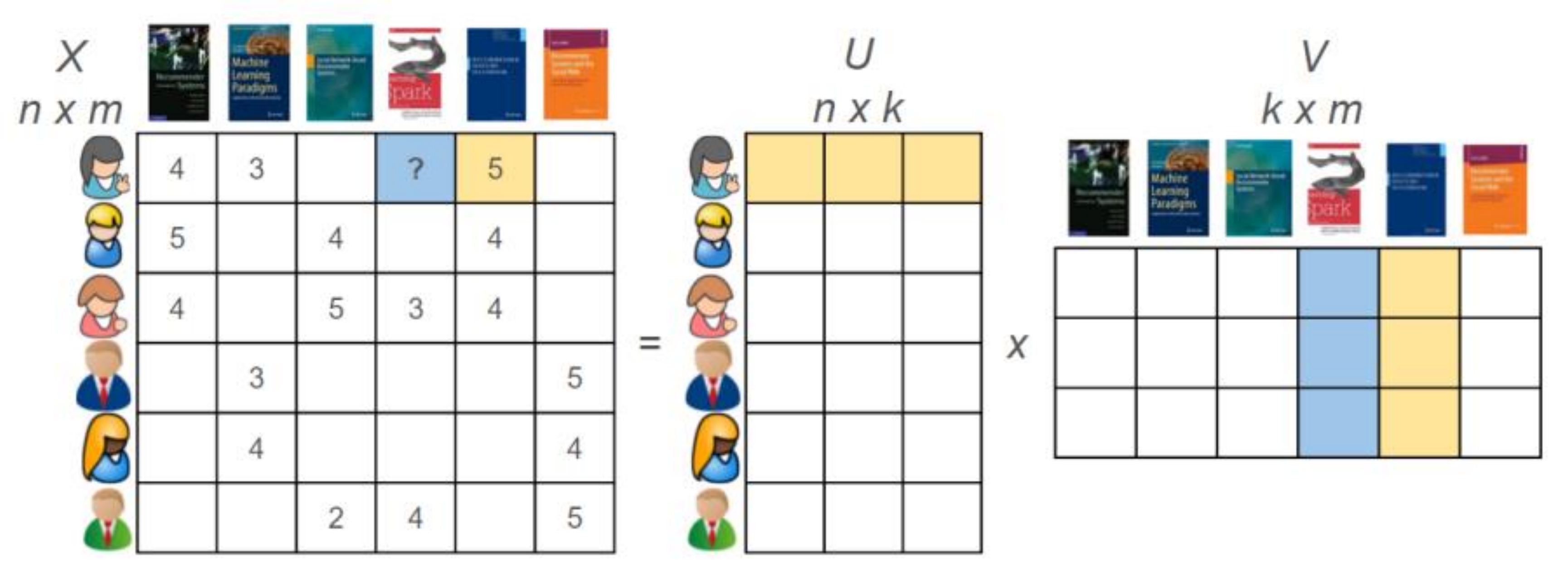
DAAL: Intel's Data Analytics Acceleration Library



DAAL is an open-source project that provides:

- Algorithms Kernels to Users
  - Batch Mode (Single Node)
  - **Distributed Mode (multi nodes)**
  - Streaming Mode (single node)
- Data Management & APIs to Developers
  - Data structure, e.g., Table, Map, etc.
  - HPC Kernels and Tools: MKL, TBB, etc.
  - Hardware Support: Compiler

Case Study: Matrix-Factorization Based on SGD (MF-SGD)



4	3		?	5	
5		4		4	
4		5	3	4	
	3				5
	4				4
		2	4		5

$$X = UV$$

$$E_{ij} = X_{ij} - \overset{r}{\underset{k=0}{}} U_{ik} V_{kj}$$

$$U_{i*}^{t} = U_{i*}^{t-1} - \eta(E_{ij}^{t-1} \cdot V_{*j}^{t-1} - \lambda \cdot U_{i*}^{t-1}$$

$$V_{*j}^{t} = V_{*j}^{t-1} - \eta(E_{ij}^{t-1} \cdot U_{i*}^{t-1} - \lambda \cdot V_{*j}^{t-1}$$

Decompose a large matrix into two model matrices, used in Recommender systems

- Large Training Data: Tens of millions of points
- Large Model Data: m, n could be millions
- Random Memory Access Pattern in Training

Stochastic Gradient Descent

The standard SGD will loop over all the nonzero ratings $x_{i,j}$ *in a random way*

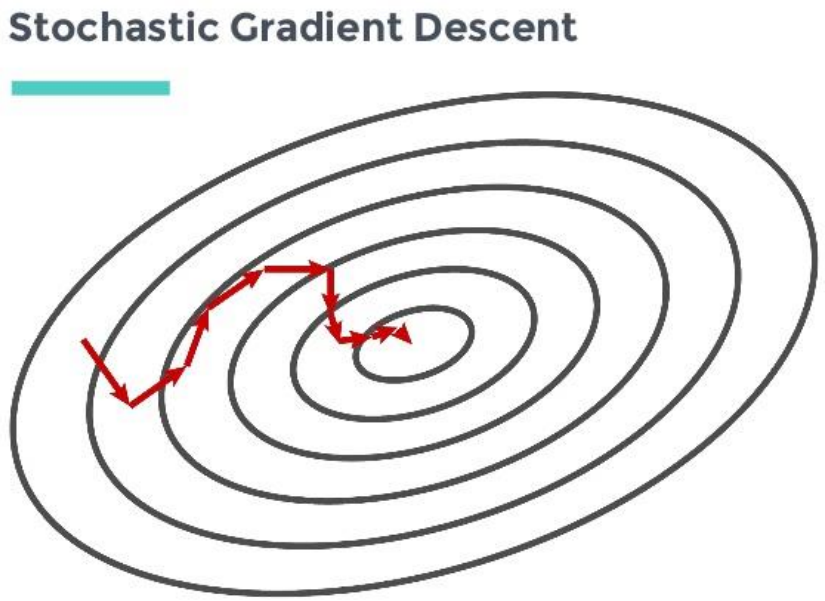
- Compute the errors

$$e_{i,j} = x_{i,j} - u_{i,*}v_{*,j}$$

- Update the factors U and V

$$u_{i,*} = u_{i,*} + \gamma \cdot (e_{i,j} \cdot v_{*,j} - \lambda \cdot u_{i,*})$$

$$v_{*,j} = v_{*,j} + \gamma \cdot (e_{i,j} \cdot u_{i,*} - \lambda \cdot v_{*,j})$$



Challenge of SGD in Big Model Problem

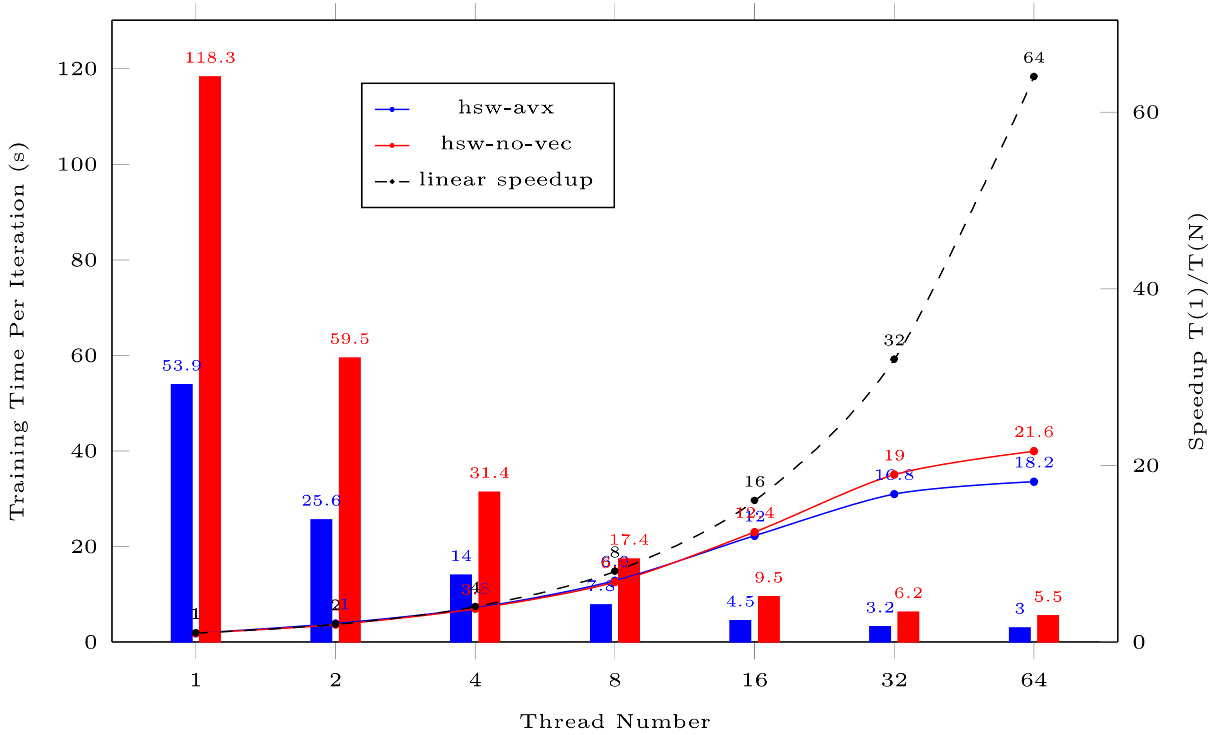
1. Memory Wall

- 4 memory ops for 3 computation ops In updating U and V

Processor is hungry of data !!

2. Random Memory Access

- Difficulty in data prefetching
- Inefficiency in using cache



Strong Scaling of SGD on Haswell CPU with Multithreading

We test a multi-threading SGD on a CPU

The strong scalability collapses after using more than 16 threads !!

What for Novel Hardware Architectures and Runtime Systems

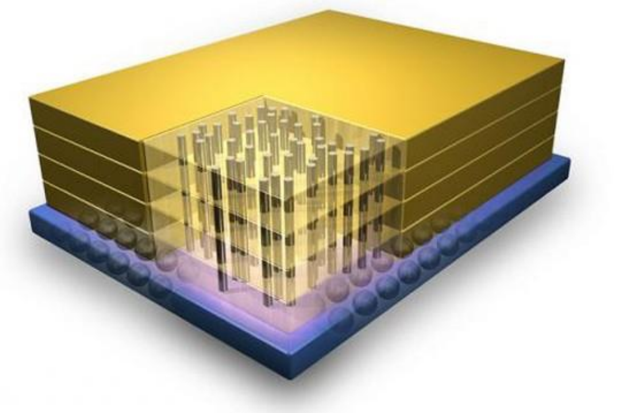
Hardware Aspect:

- 3D stack memory
- Many-core: GPU, Xeon Phi, FPGA, etc.

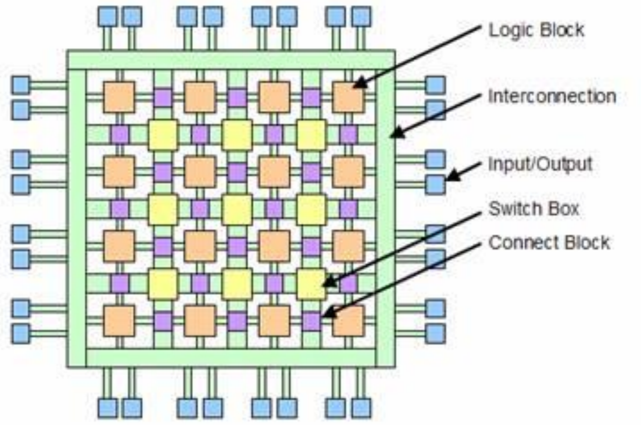
Software Aspect:

- Runtime System
- Dynamic Task Scheduling

Reduce the memory access latency
Increase memory bandwidth

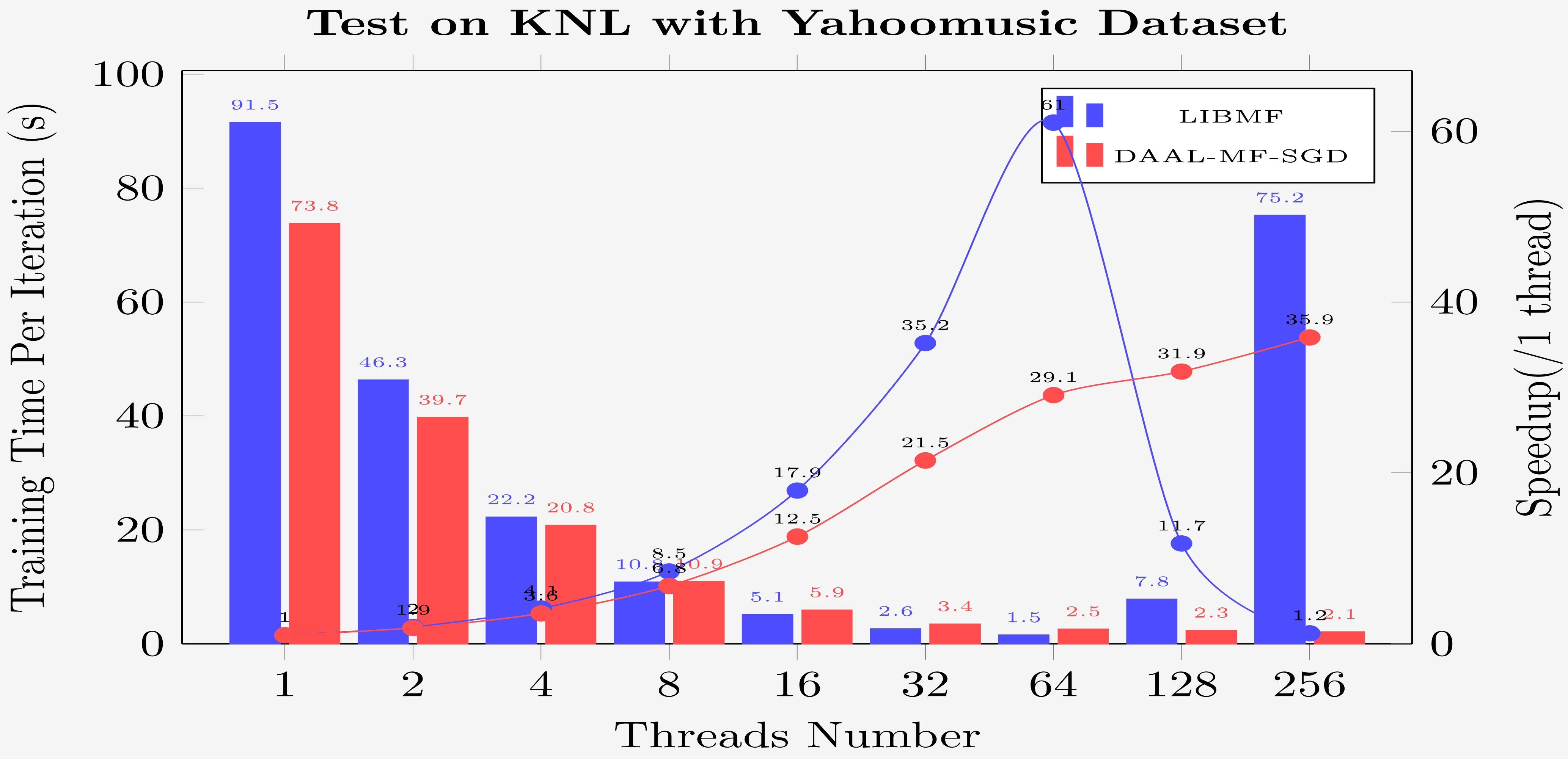


IBM and Micron's big memory cube



A generalized architecture for an FPGA

Intra-node Performance: DAAL-MF-SGD vs. LIBMF

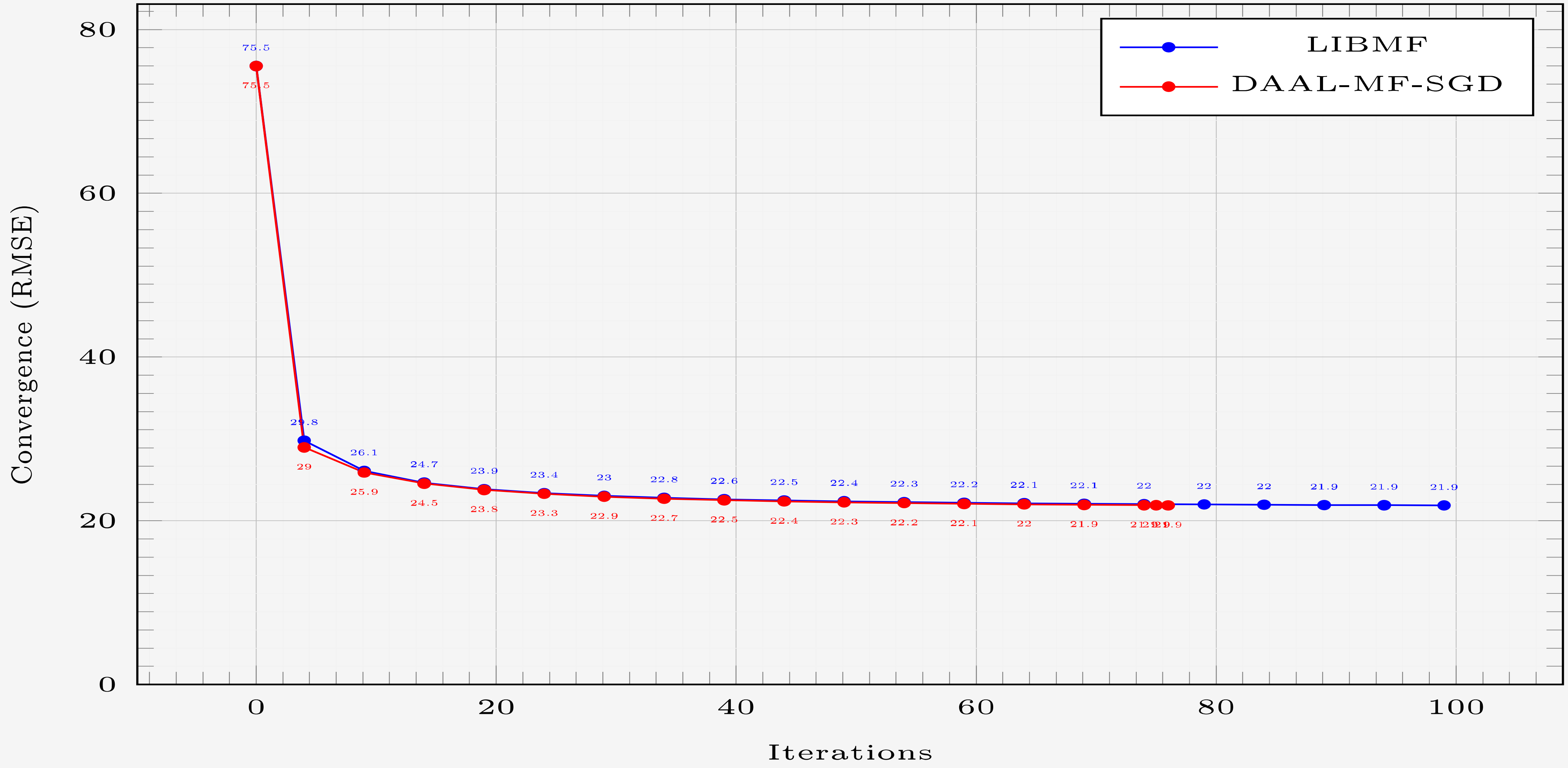


LIBMF: a start-of-art open source MF-SGD package

- Only single node mode
- Highly optimized for memory usage

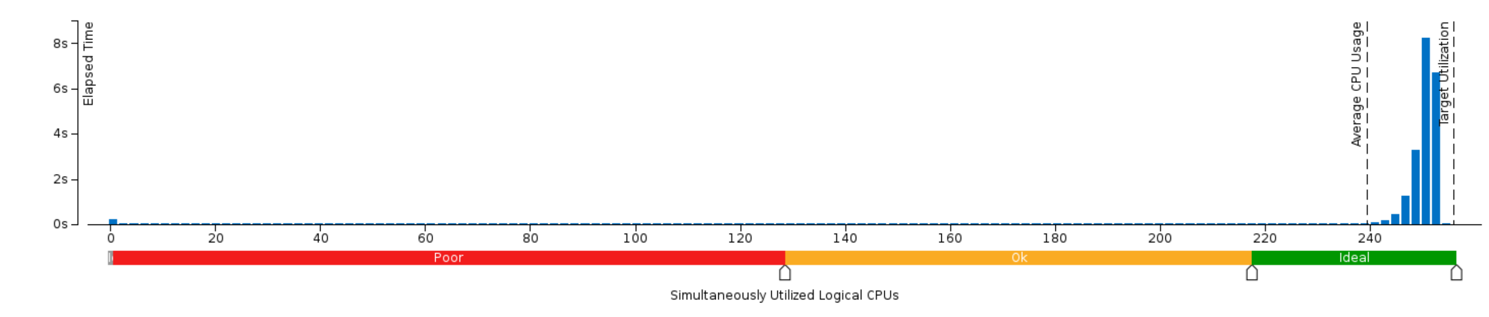
We compare our DAAL-MF-SGD kernel with LIBMF on a single KNL node, using YahooMusic dataset

Intra-node Performance: DAAL-MF-SGD vs. LIBMF

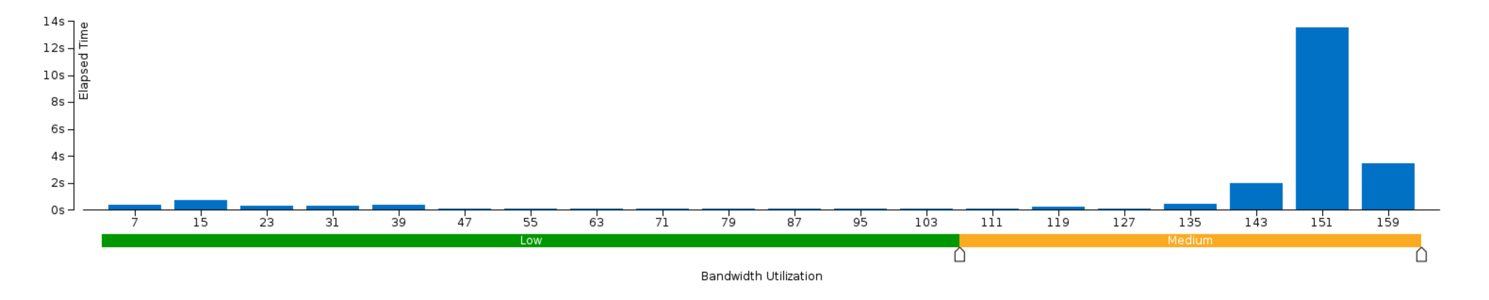


- DAAL-MF-SGD delivers a comparable training time for each iteration with that of LIBMF
- DAAL-MF-SGD has a better convergence speed than LIBMF, using less iterations to achieve the same convergence.

CPU utilization and Memory Bandwidth on KNL



CPU (threads) utilization on KNL



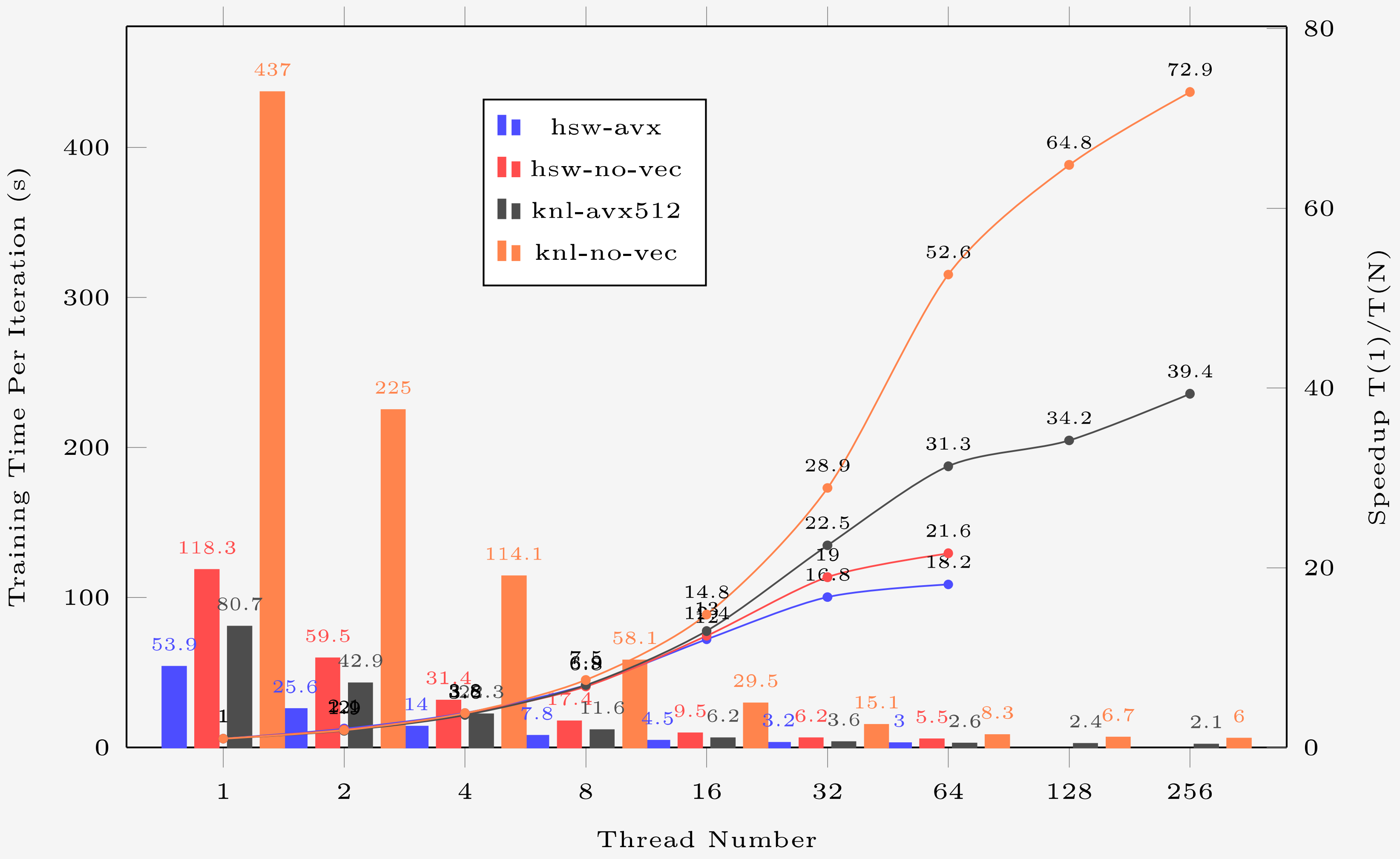
Memory Bandwidth Usage on KNL

- DAAL-MF-SGD utilizes more than 95% of all the 256 threads on KNL

- DAAL-MF-SGD uses more than half of the total bandwidth of MCDRAM on KNL

- We need to explore the full usage of all of MCDRAM's bandwidth (around 400 GB) to further speed up DAAL-MF-SGD

Intra-node Performance: Haswell Xeon vs. KNL Xeon Phi



DAAL-MF-SGD has a better performance on KNL than on Haswell CPU, because it benefits from

- KNL's AVX512 vectorization
- High Memory Bandwidth

KNL has

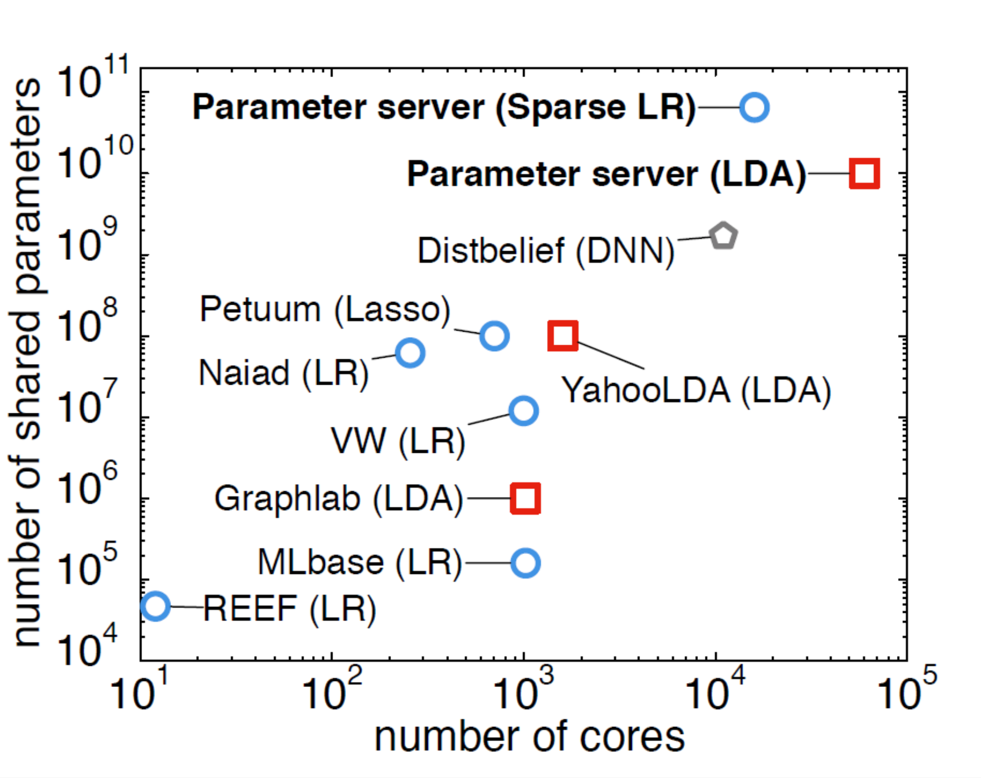
- **3x speeds up** by vectorization
- **1.5x – 4x speeds up** to Haswell

SALSA

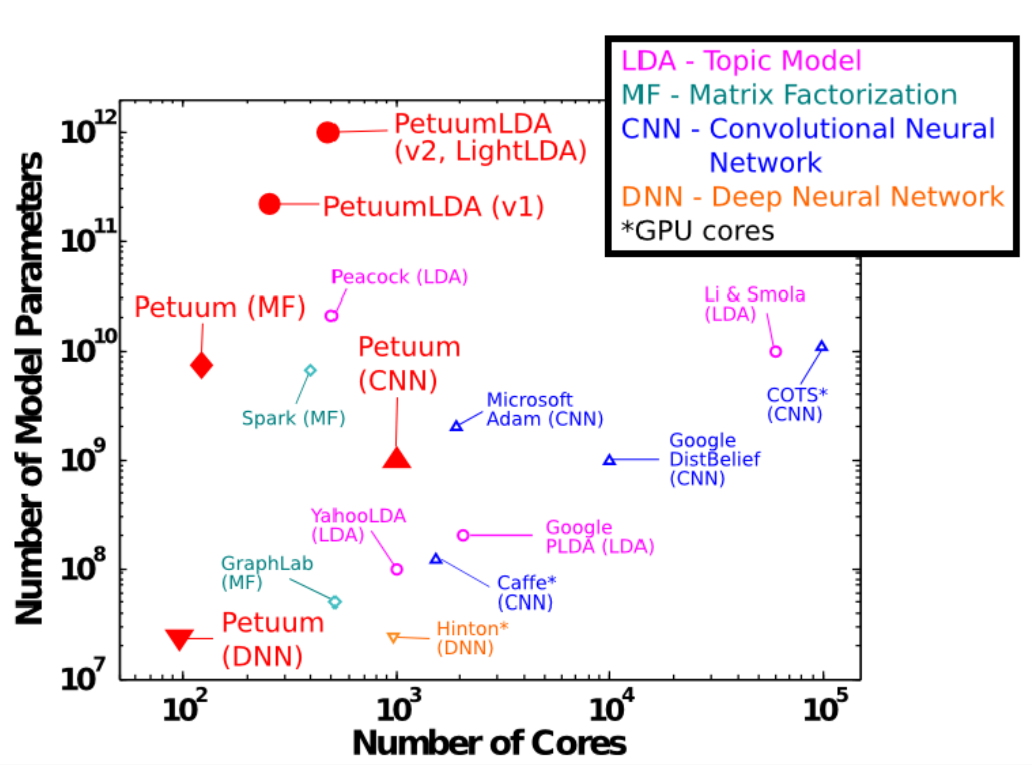
Machine Learning using Harp Framework

The Growth of Model Sizes and Scales of Machine Learning Applications

2014



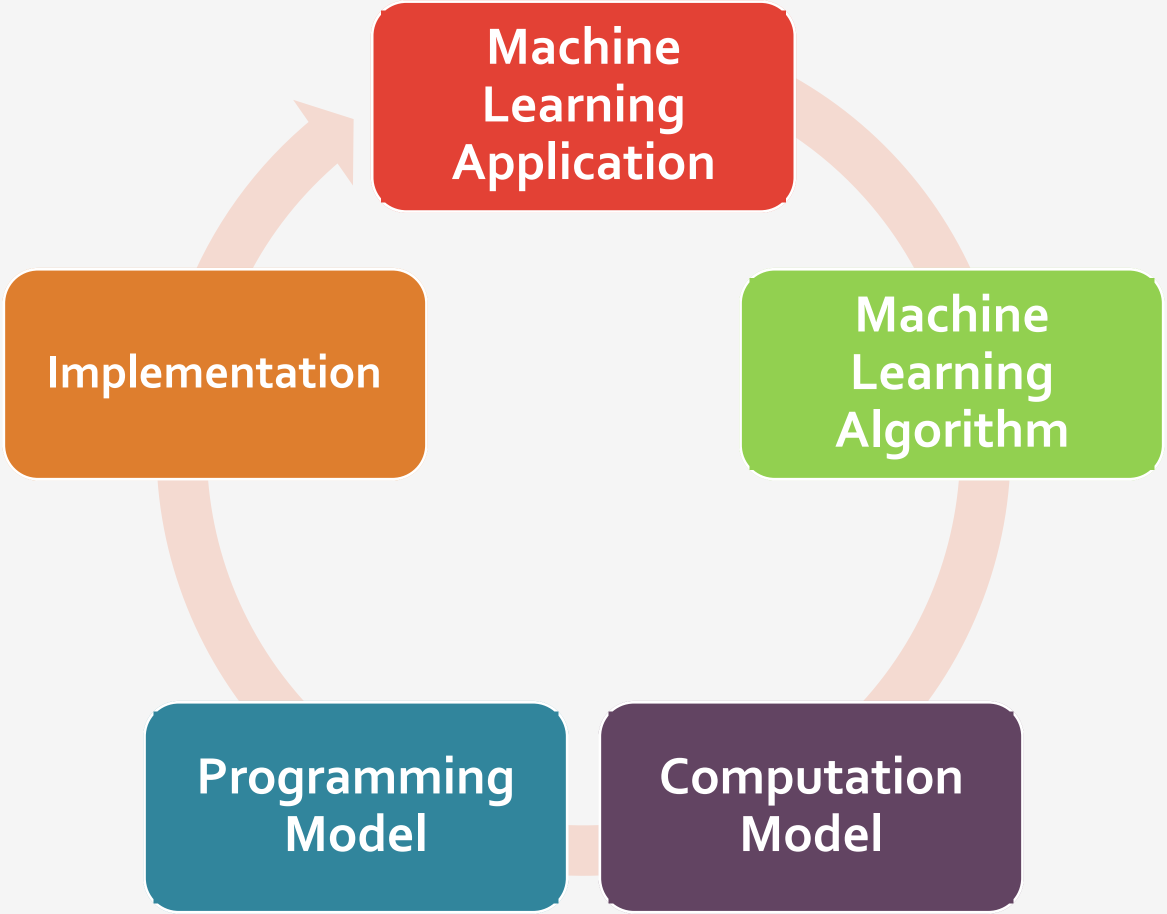
2015



Challenges of Parallelization Machine Learning Applications

- Big training data
- Big model
- Iterative computation, both CPU-bound and memory-bound
- High frequencies of model synchronization

Parallelizing Machine Learning Applications



Types of Machine Learning Applications and Algorithms

Expectation-Maximization Type

- K-Means Clustering
- Collapsed Variational Bayesian for topic modeling (e.g. LDA)

Gradient Optimization Type

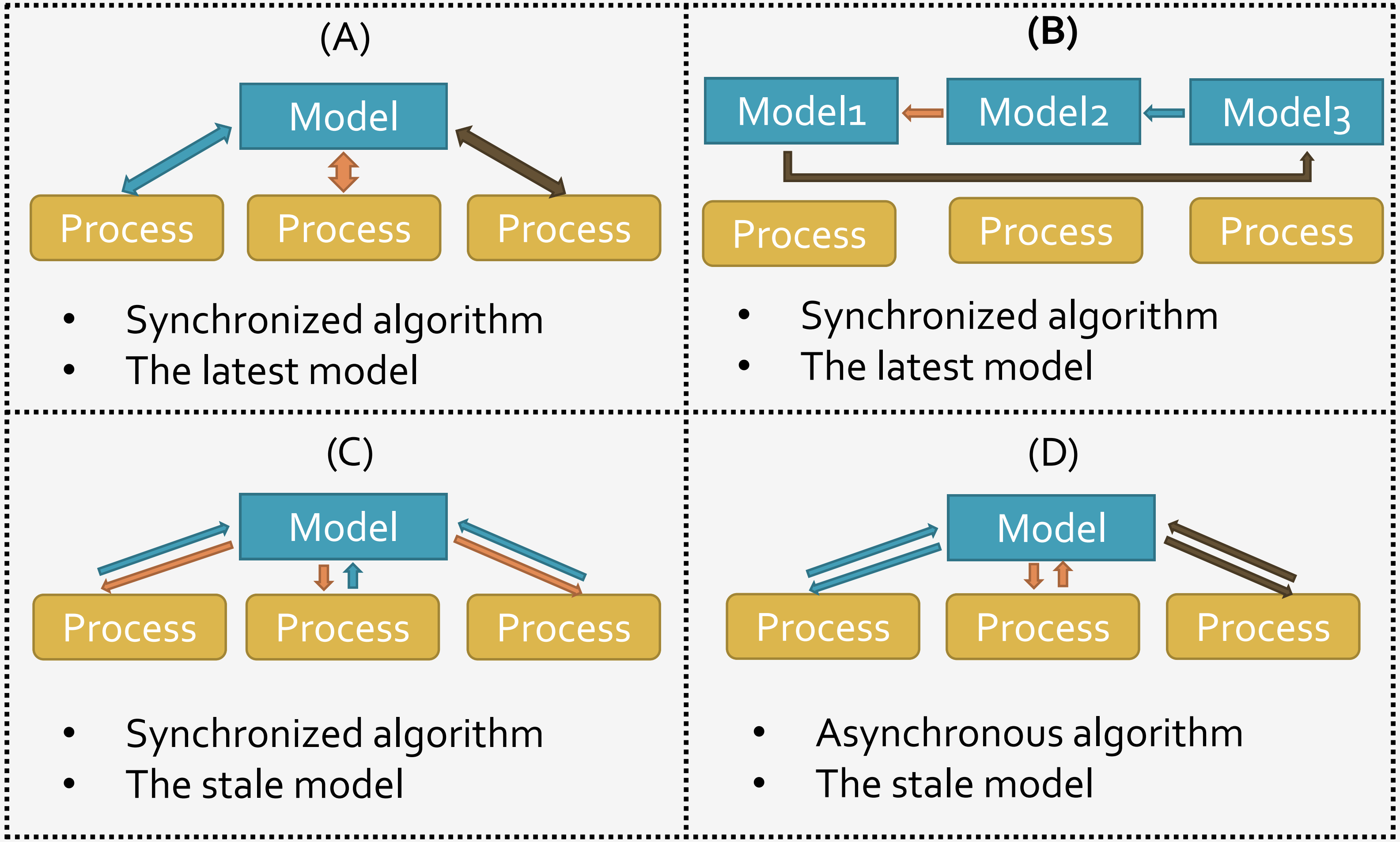
- Stochastic Gradient Descent and Cyclic Coordinate Descent for classification (e.g. SVM and Logistic Regression), regression (e.g. LASSO), collaborative filtering (e.g. Matrix Factorization)

Markov Chain Monte Carlo Type

- Collapsed Gibbs Sampling for topic modeling (e.g. LDA)

Inter/Intra-node Computation Models

Model-Centric Synchronization Paradigms



(A)

- Synchronized algorithm
- The latest model

(B)

- Synchronized algorithm
- The latest model

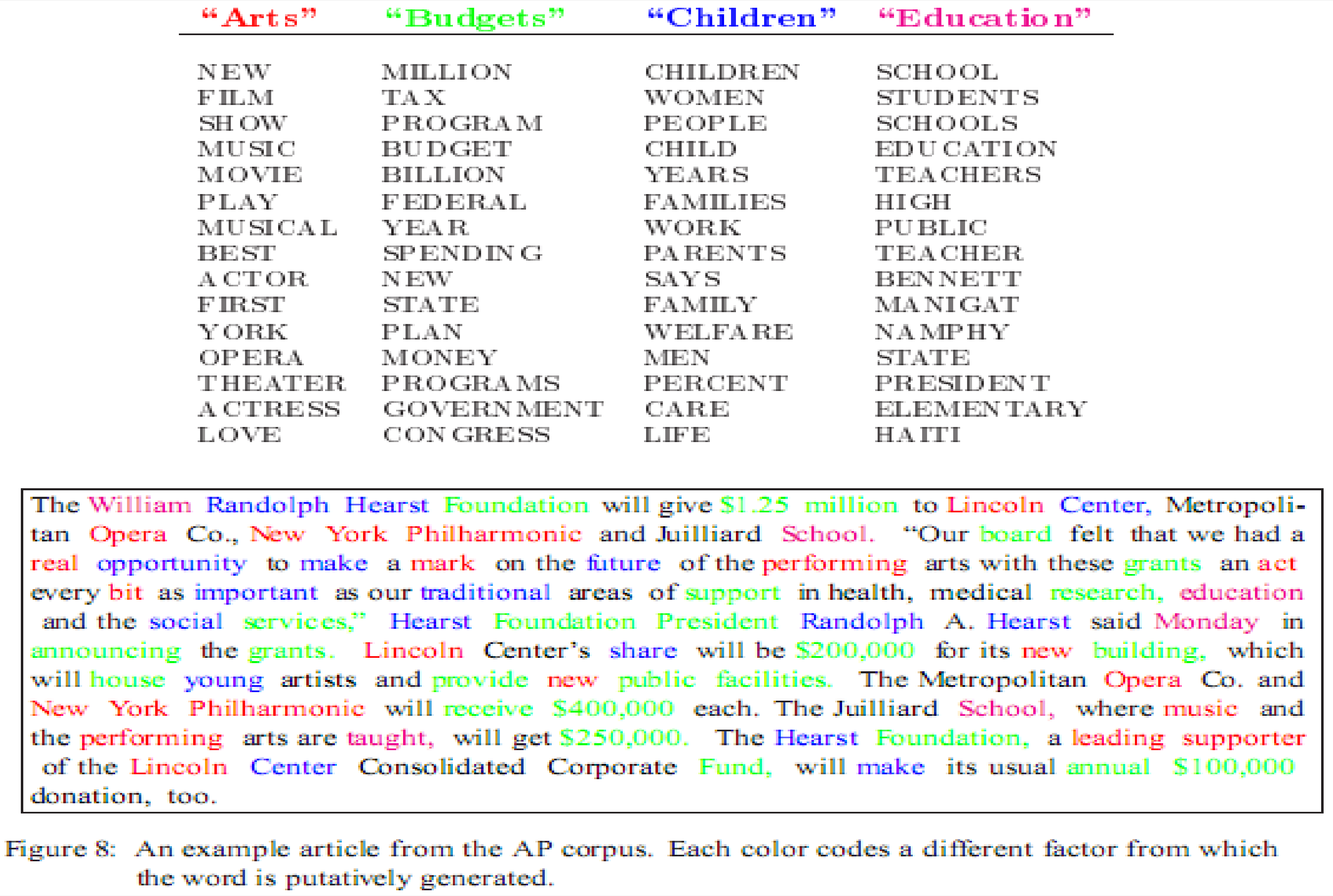
(C)

- Synchronized algorithm
- The stale model

(D)

- Asynchronous algorithm
- The stale model

Case Study: LDA mines topics in text collection



“Arts”	“Budgets”	“Children”	“Education”
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

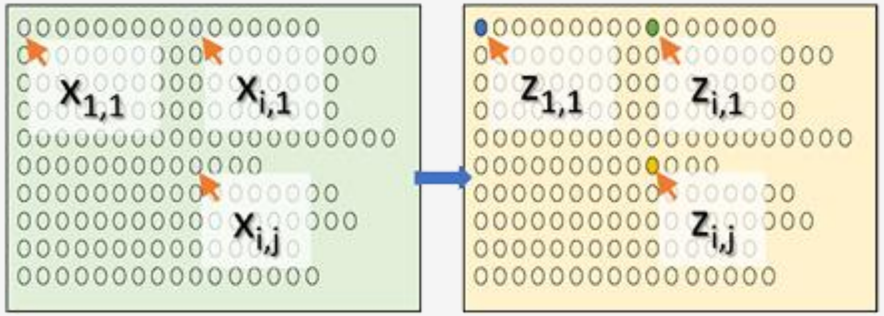
Figure 8: An example article from the AP corpus. Each color codes a different factor from which the word is putatively generated.

- Huge volume of Text Data
  - Information overloading
  - What on earth is inside the TEXT Data?
- Search
  - Find the documents relevant to my need (ad hoc query)
- Filtering
  - Fixed info needs and dynamic text data
- What's new inside?
  - Discover something I don't know

Blei, D. M., Ng, A. Y. & Jordan, M. I. Latent Dirichlet Allocation. J. Mach. Learn. Res. 3, 993–1022 (2003).

LDA and Topic model

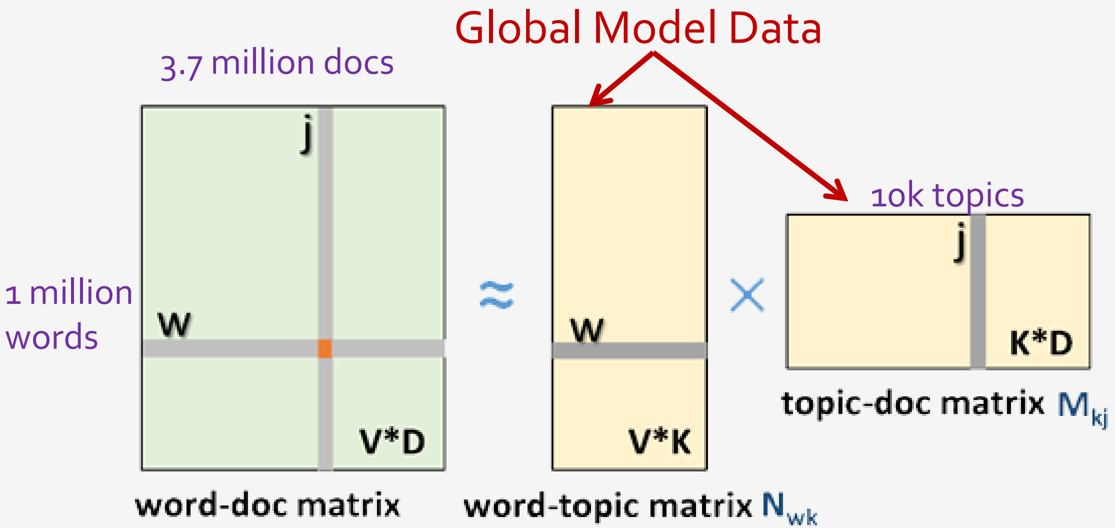
- Topic Models is a modeling technique, modeling the data by probabilistic generative process.
- Latent Dirichlet Allocation (LDA) is one widely used topic model.
- Inference algorithm for LDA is an iterative algorithm using share global model data.



Document Collection

Topic assignment

- Document
- Word
- Topic: semantic unit inside the data
- Topic Model
  - documents are mixtures of topics, where a topic is a probability distribution over words



Global Model Data

3.7 million docs

1 million words

10k topics

topic-doc matrix M_{kj}

word-doc matrix

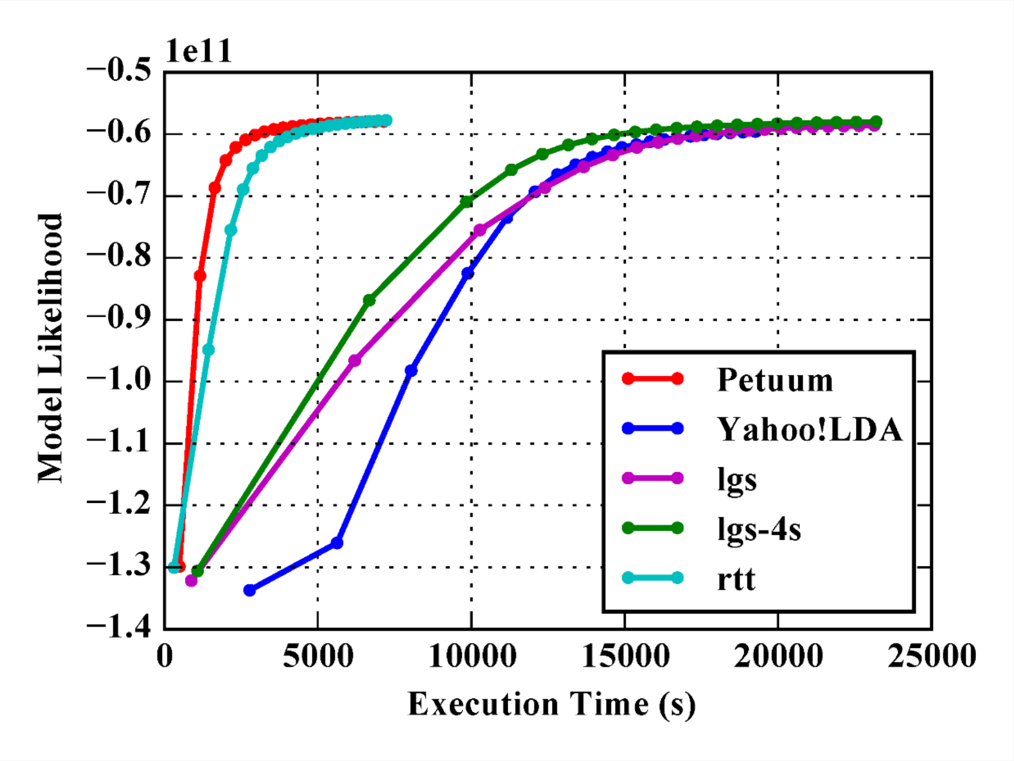
word-topic matrix N_{wk}

Normalized co-occurrence matrix

Mixture components

Mixture weights

The Comparison of LDA CGS Model Convergence Speed



rtt & Petuum: rotate model parameters
lgs & lgs-4s: one or more rounds of model synchronization per iteration
Yahoo!LDA: asynchronously fetch model parameters

The parallelization strategy can highly affect the algorithm convergence and the system efficiency. This brings us four questions:

- **What part of the model needs to be synchronized?** The parallelization needs to decide which model parts needs synchronization.
- **When should the model synchronization happen?** In the parallel execution timeline, the parallelization should choose the time point to perform model synchronization.
- **Where should the model synchronization occur?** The parallelization needs to tell the distribution of the model among parallel components, what parallel components are involved in the model synchronization.
- **How is the model synchronization performed?** The parallelization needs to explain the abstraction and the mechanism of the model synchronization.

Inter-node Computation Models

(Training Data Items Are Partitioned to Each Process)

Computation Model A

- Once a process trains a data item, it locks the related model parameters and prevents other processes from accessing them. When the related model parameters are updated, the process unlocks the parameters. Thus the model parameters used in local computation are always the latest.

Computation Model B

- Each process first takes a part of the shared model and performs training. Afterwards, the model is shifted between processes. Through model rotation, each model parameters are updated by one process at a time so that the model is consistent.

Computation Model C

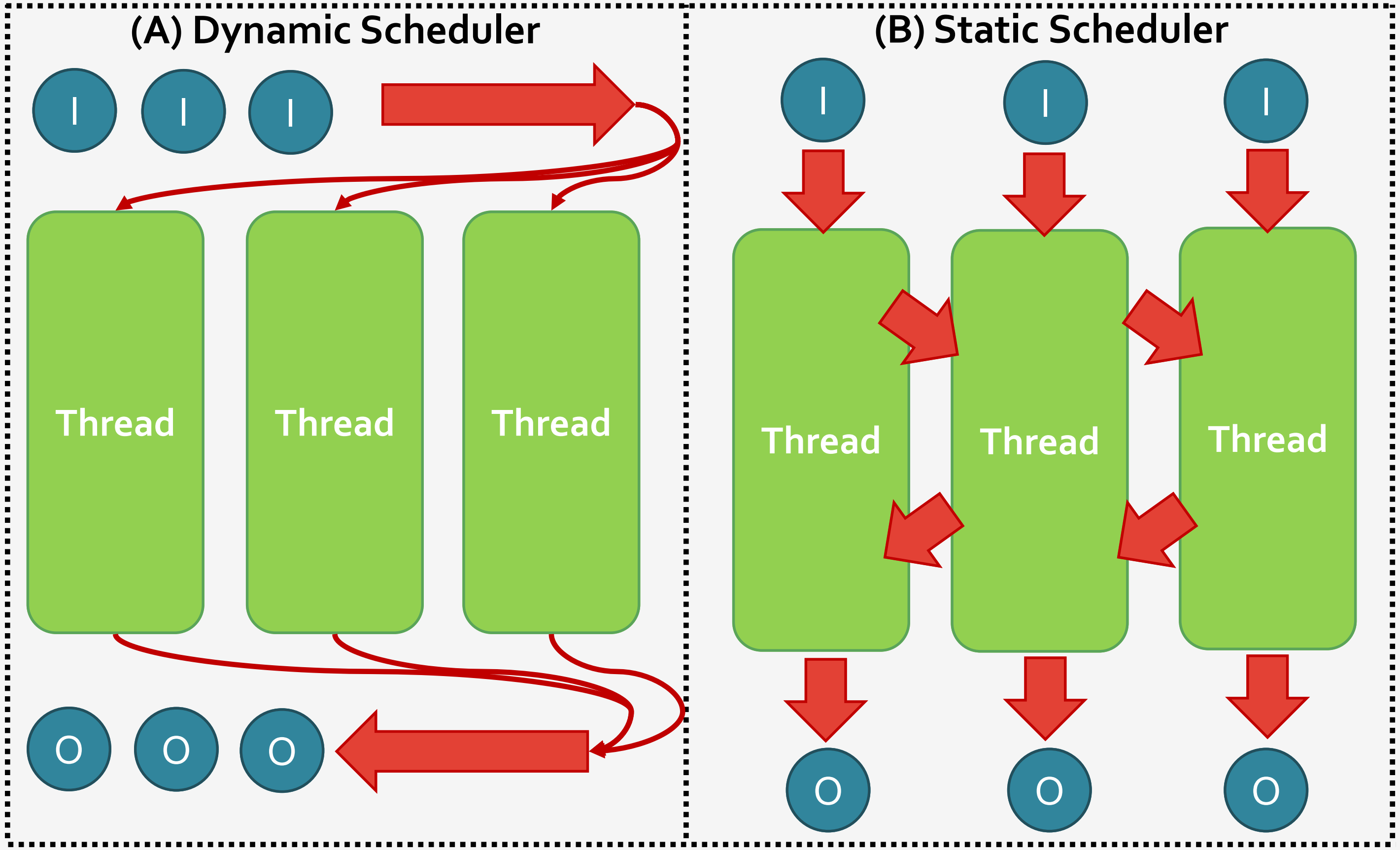
- Each process first fetches all the model parameters required by local computation. When the local computation is completed, modifications of the local model from all processes are gathered to update the model.

Computation Model D

- Each process independently fetches related model parameters, performs local computation, and returns model modifications. Unlike A, workers are allowed to fetch or update the same model parameters in parallel. In contrast to B and C, there is no synchronization barrier.

Intra-node: Schedule Data Partitions to Threads

(only Data Partitions in Computation Model A, C, D; Data and/or Model Partitions in B)



(A) Dynamic Scheduler

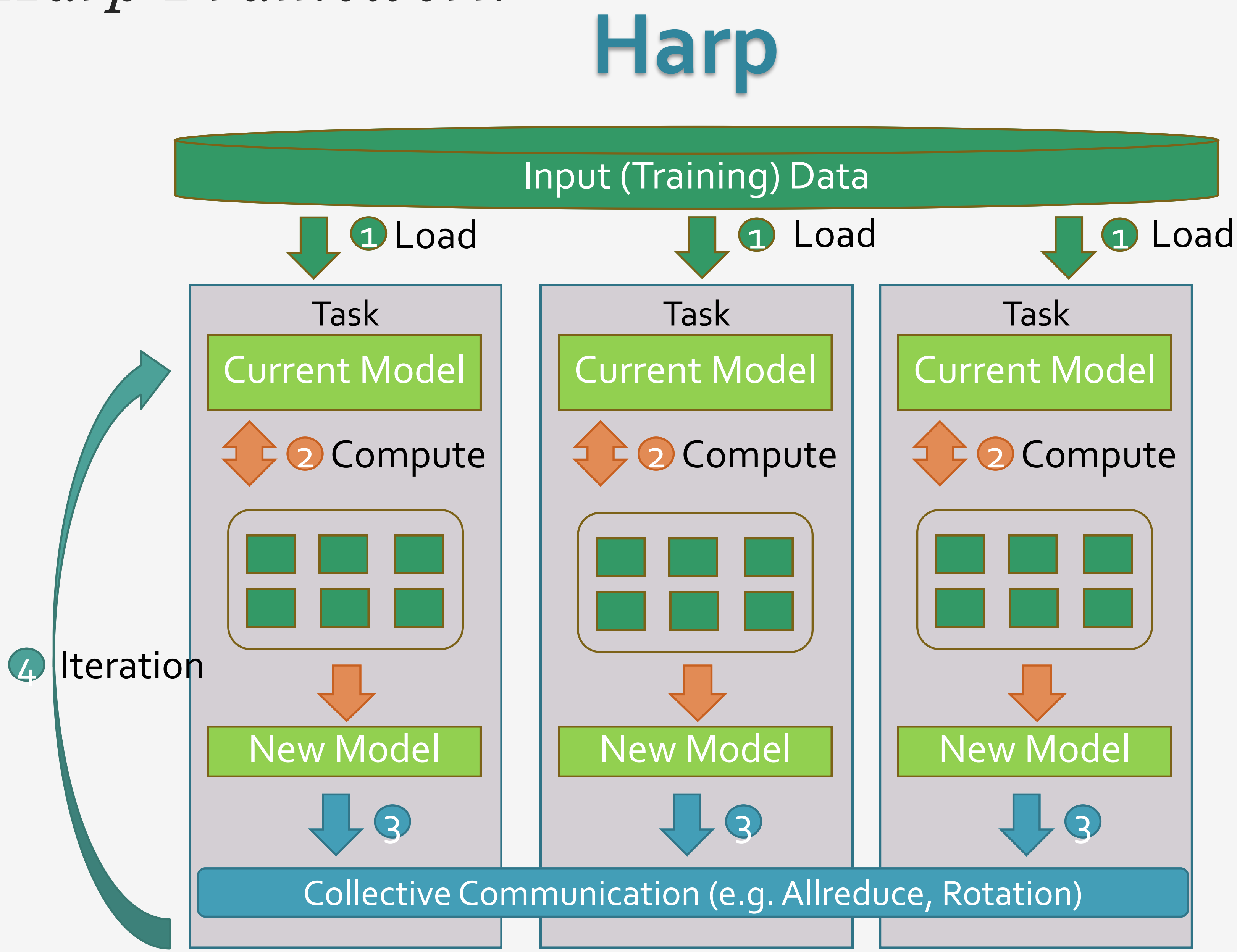
- All computation models can use this scheduler.
- All the inputs are submitted to one queue.
- Threads dynamically fetch inputs from the queue.
- The main thread can retrieve the outputs from the output queue.

(B) Static Scheduler

- All computation models can use this scheduler.
- Each thread has its own input queue and output queue.
- Each thread can submit inputs to another thread .
- The main thread can retrieve outputs from each task's output queue.

SALSA

Harp Framework

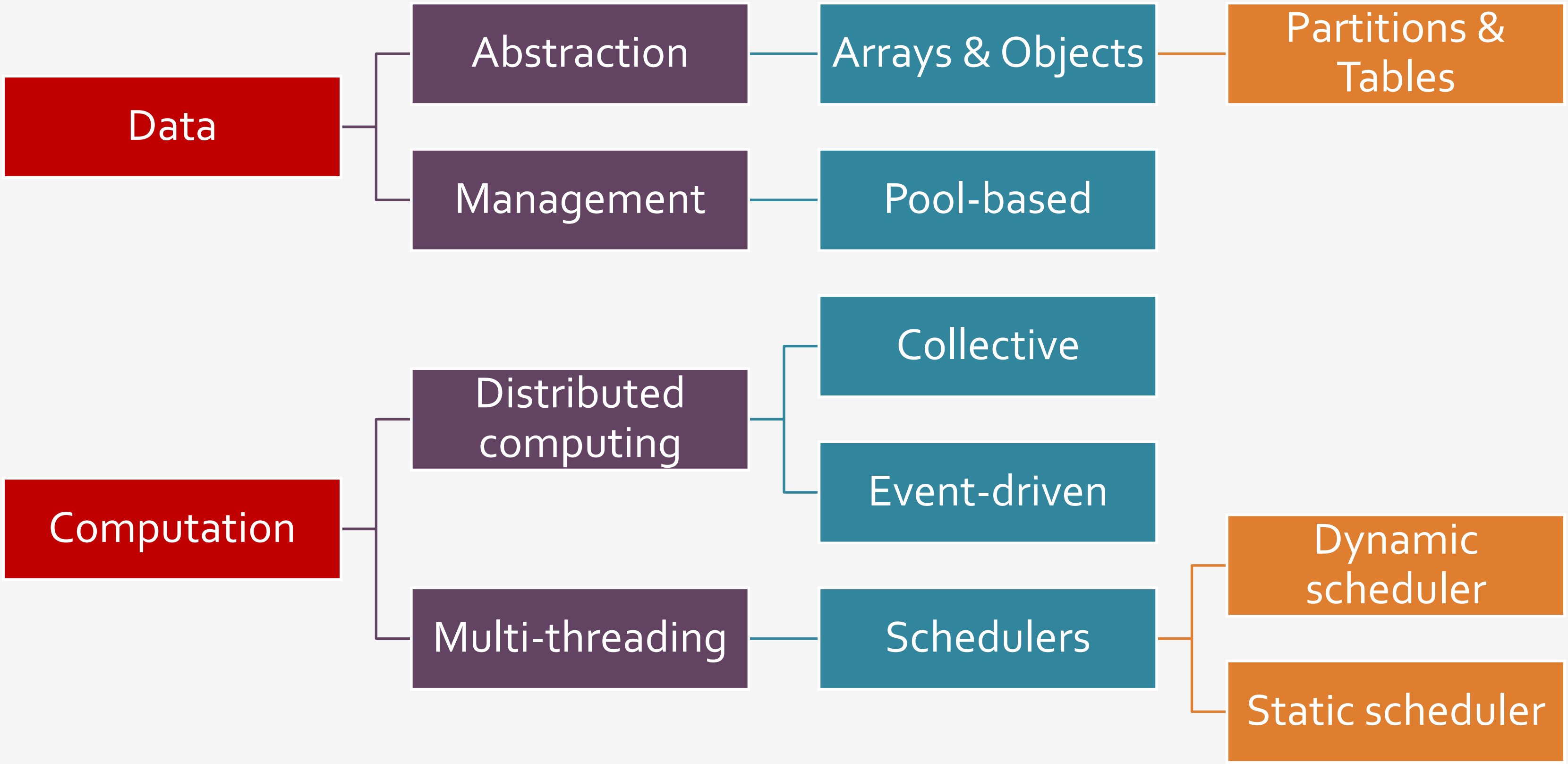


Harp is an open-source project developed by
Indiana University.

- MPI-like collective communication operations that are highly optimized for big data problems.
- Harp has efficient and innovative computation models for different machine learning problems.

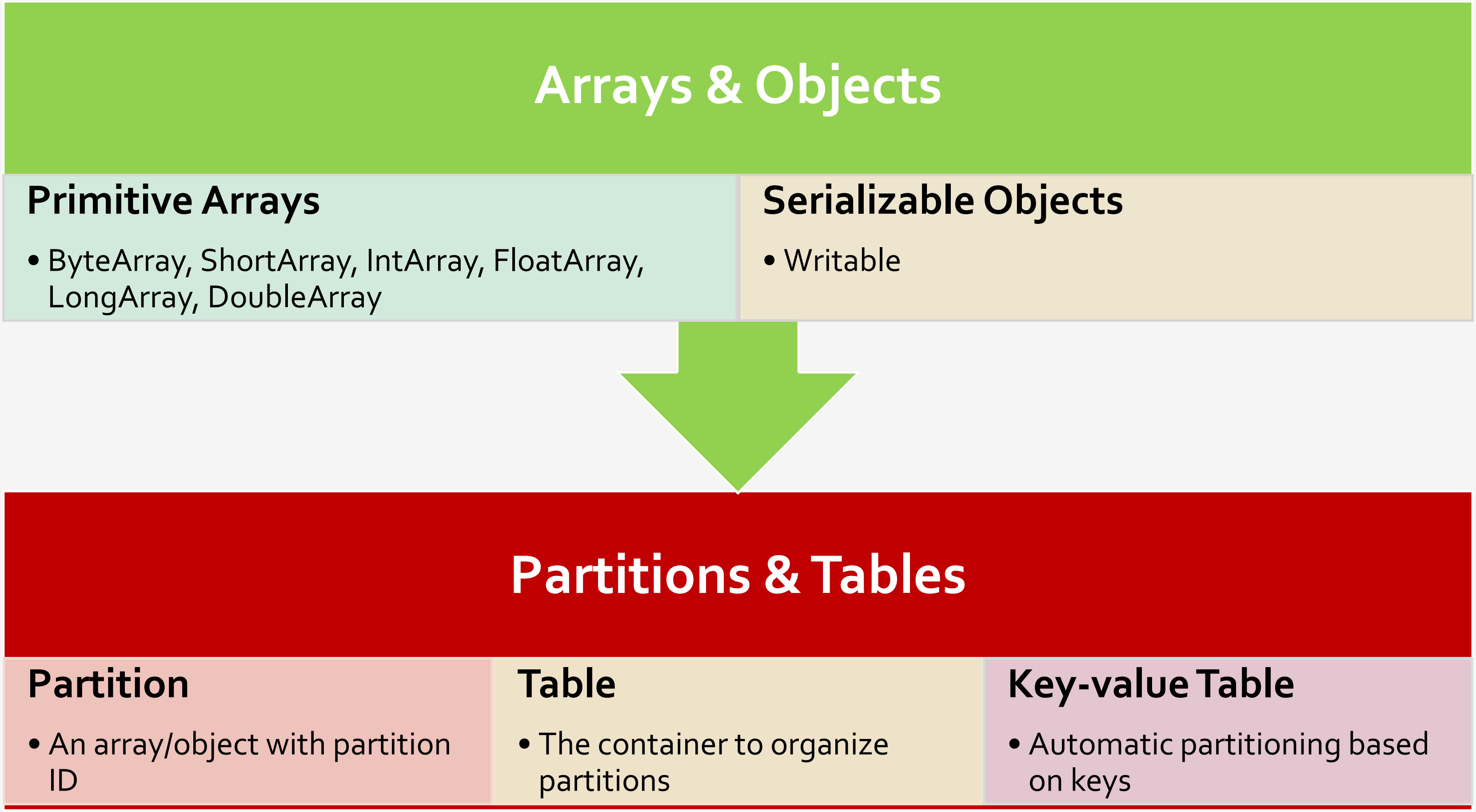
SALSA

Harp Features



SALSA

Data Types



Arrays & Objects

Primitive Arrays

- ByteArray, ShortArray, IntArray, FloatArray, LongArray, DoubleArray

Serializable Objects

- Writable

Partitions & Tables

Partition

- An array/object with partition ID

Table

- The container to organize partitions

Key-value Table

- Automatic partitioning based on keys

APIs

Scheduler

- DynamicScheduler
- StaticScheduler

Collective

- broadcast
- reduce
- allgather
- allreduce
- regroup
- pull
- push
- rotate

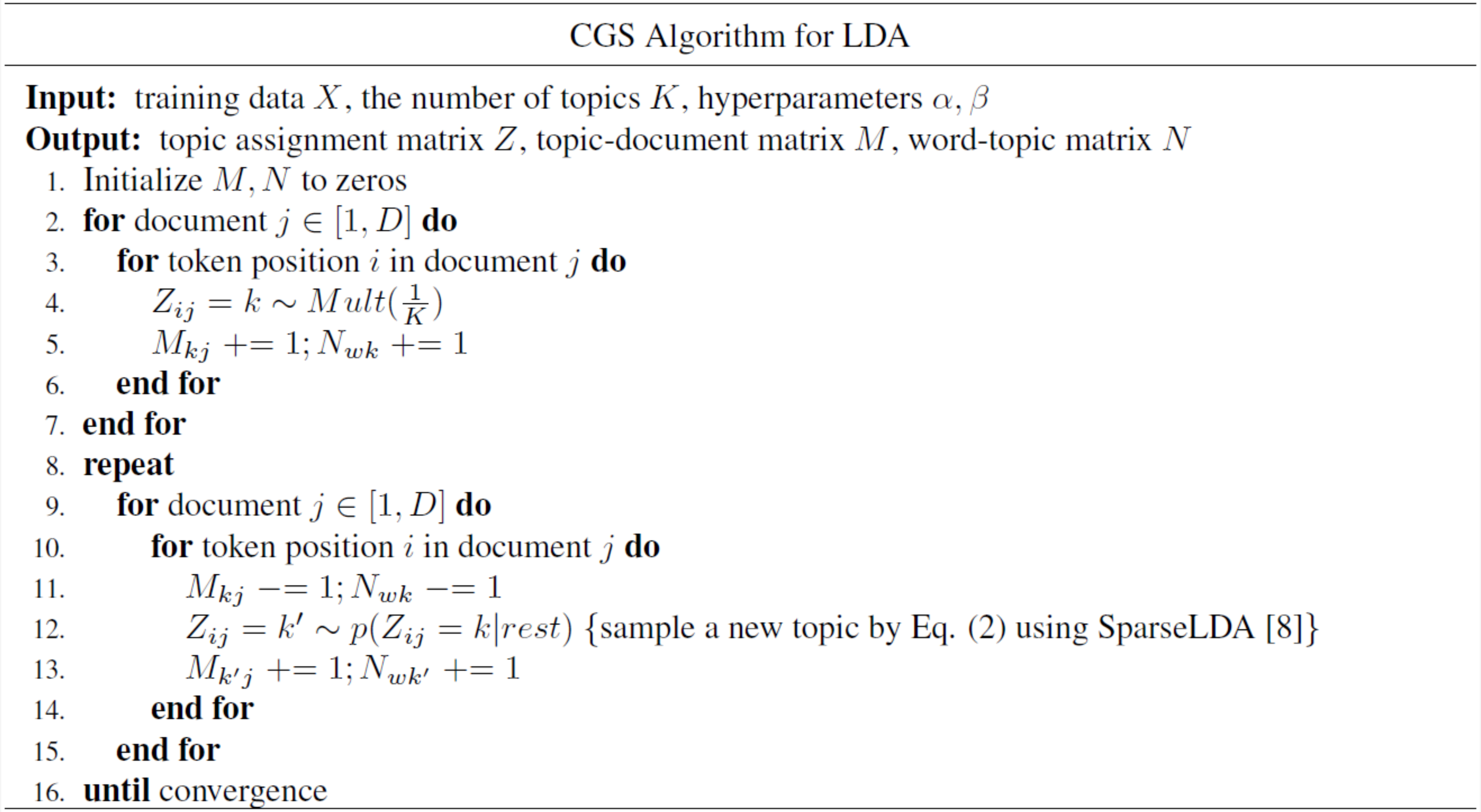
Event Driven

- getEvent
- waitEvent
- sendEvent

Case Study: LDA and Matrix-Factorization Based on SGD and CCD

Dataset	Node Type	
	Xeon E5 2699 v3 (each uses 30 Threads)	Xeon E5 2670 v3 (each uses 20 Threads)
clueweb1	Harp CGS vs. Petuum (30)	Harp CGS vs. Petuum (60)
clueweb2	Harp SGD vs. NOMAD (30) Harp CCD vs. CCD++ (30)	Harp SGD vs. NOMAD (60) Harp CCD vs. CCD++ (60)

Collapsed Gibbs Sampling for Latent Dirichlet Allocation



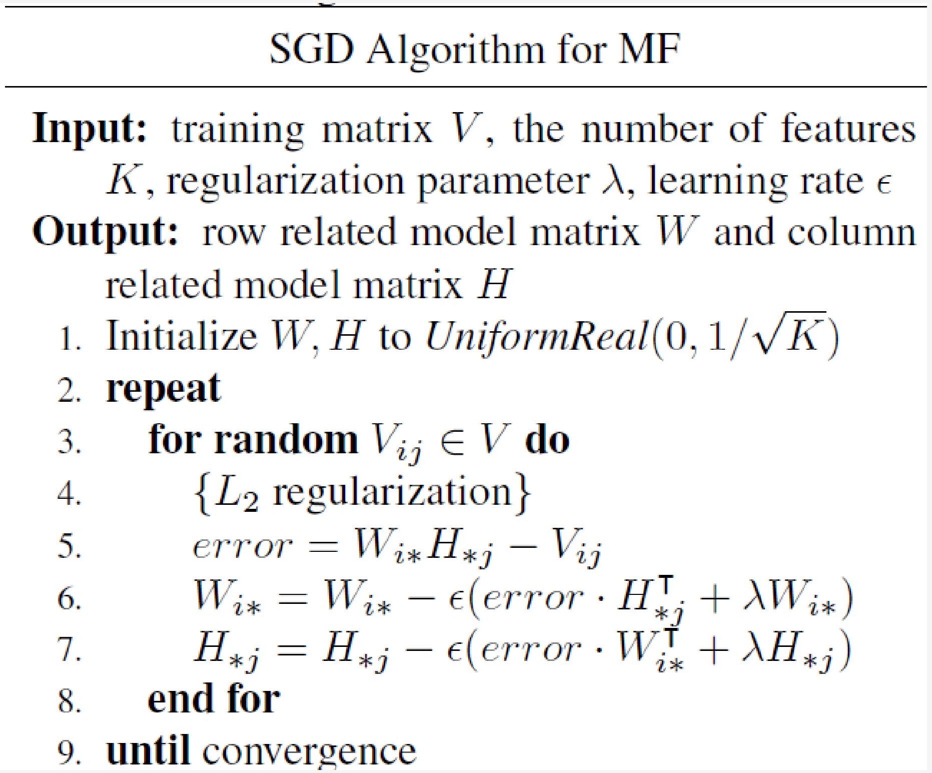
CGS Algorithm for LDA

Input: training data X, the number of topics K, hyperparameters α, β

Output: topic assignment matrix Z, topic-document matrix M, word-topic matrix N

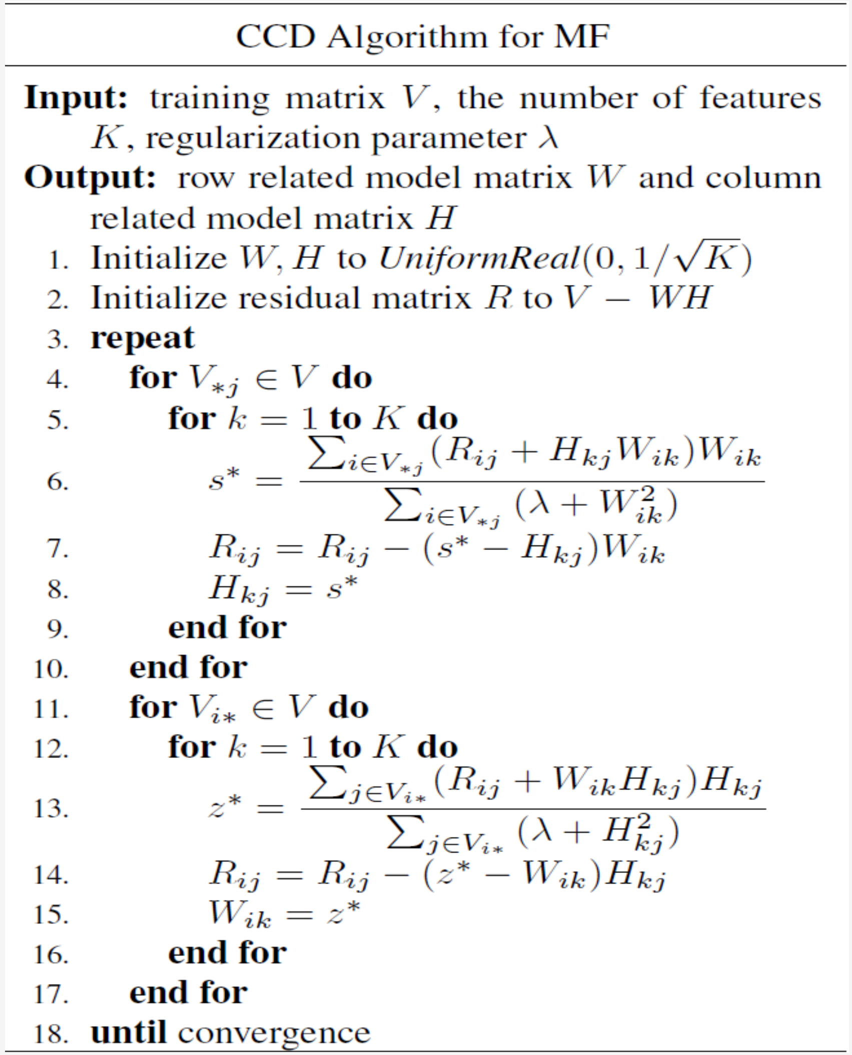
```
Initialize $M, N$ to zeros
for document $j \in [1, D]$ do
    for token position $i$ in document $j$ do
        $Z_{ij} = k \sim Mult(\frac{1}{K})$
        $M_{kj}$ += 1; $N_{wk}$ += 1
    end for
end for
repeat
    for document $j \in [1, D]$ do
        for token position $i$ in document $j$ do
            $M_{kj}$ -= 1; $N_{wk}$ -= 1
            $Z_{ij} = k' \sim p(Z_{ij} = k|rest)$ {sample a new topic by Eq. (2) using SparseLDA [8]}
            $M_{k'j}$ += 1; $N_{wk'}$ += 1
        end for
    end for
until convergence
```

Matrix Factorization



SGD Algorithm for MF

```
Input: training matrix $V$, the number of features $K$, regularization parameter $\lambda$, learning rate $\epsilon$
Output: row related model matrix $W$ and column related model matrix $H$
Initialize $W, H$ to $\mathit{UniformReal}(0, 1/\sqrt{K})$
repeat
    for random $V_{ij} \in V$ do
        {$L_2$ regularization}
        $error = W_{i*}H_{*j} - V_{ij}$
        $W_{i*} = W_{i*} - \epsilon(error \cdot H_{*j}^{\intercal} + \lambda W_{i*})$
        $H_{*j} = H_{*j} - \epsilon(error \cdot W_{i*}^{\intercal} + \lambda H_{*j})$
    end for
until convergence
```



CCD Algorithm for MF

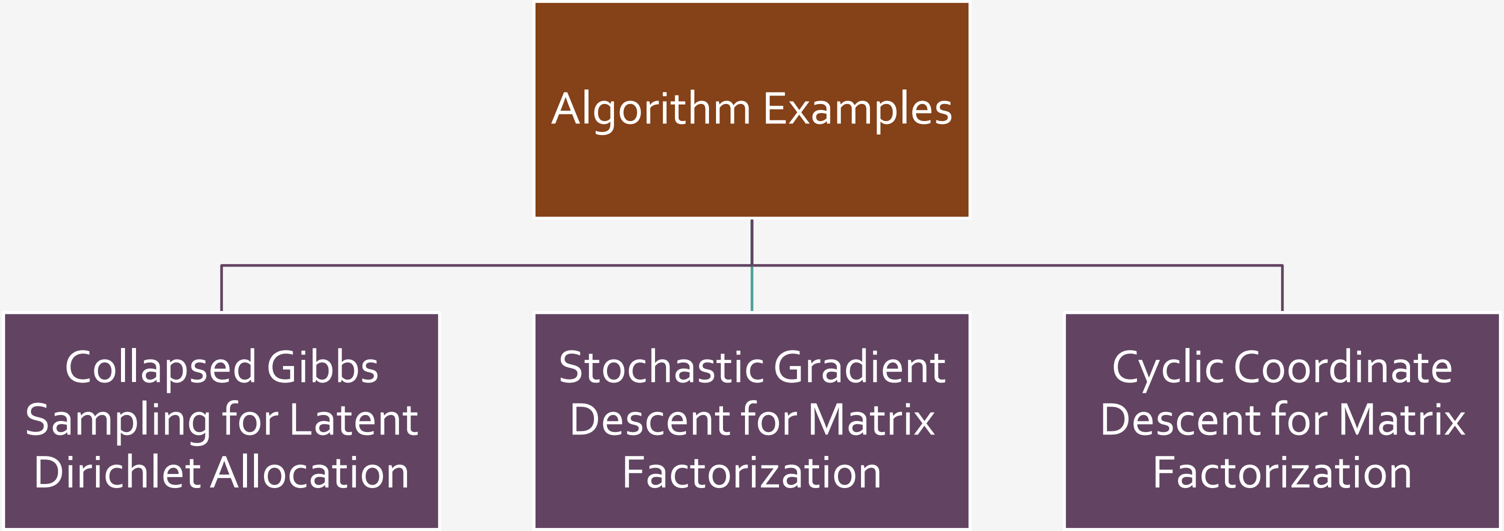
```
Input: training matrix $V$, the number of features $K$, regularization parameter $\lambda$
Output: row related model matrix $W$ and column related model matrix $H$
Initialize $W, H$ to $\mathit{UniformReal}(0, 1/\sqrt{K})$
Initialize residual matrix $R$ to $V - WH$
repeat
    for $V_{*j} \in V$ do
        for $k = 1$ to $K$ do
            $s^* = \frac{\sum_{i \in V_{*j}} (R_{ij} + H_{kj}W_{ik})W_{ik}}{\sum_{i \in V_{*j}} (\lambda + W_{ik}^2)}$
            $R_{ij} = R_{ij} - (s^* - H_{kj})W_{ik}$
            $H_{kj} = s^*$
        end for
    end for
    for $V_{i*} \in V$ do
        for $k = 1$ to $K$ do
            $z^* = \frac{\sum_{j \in V_{i*}} (R_{ij} + W_{ik}H_{kj})H_{kj}}{\sum_{j \in V_{i*}} (\lambda + H_{kj}^2)}$
            $R_{ij} = R_{ij} - (z^* - W_{ik})H_{kj}$
            $W_{ik} = z^*$
        end for
    end for
until convergence
```

Features of Model Update in Machine Learning Algorithms

I. The algorithms can converge even when the consistency of a model is not guaranteed to some extent.

II. The update order of the model parameters is exchangeable.

III. The model parameters for update can be randomly selected.



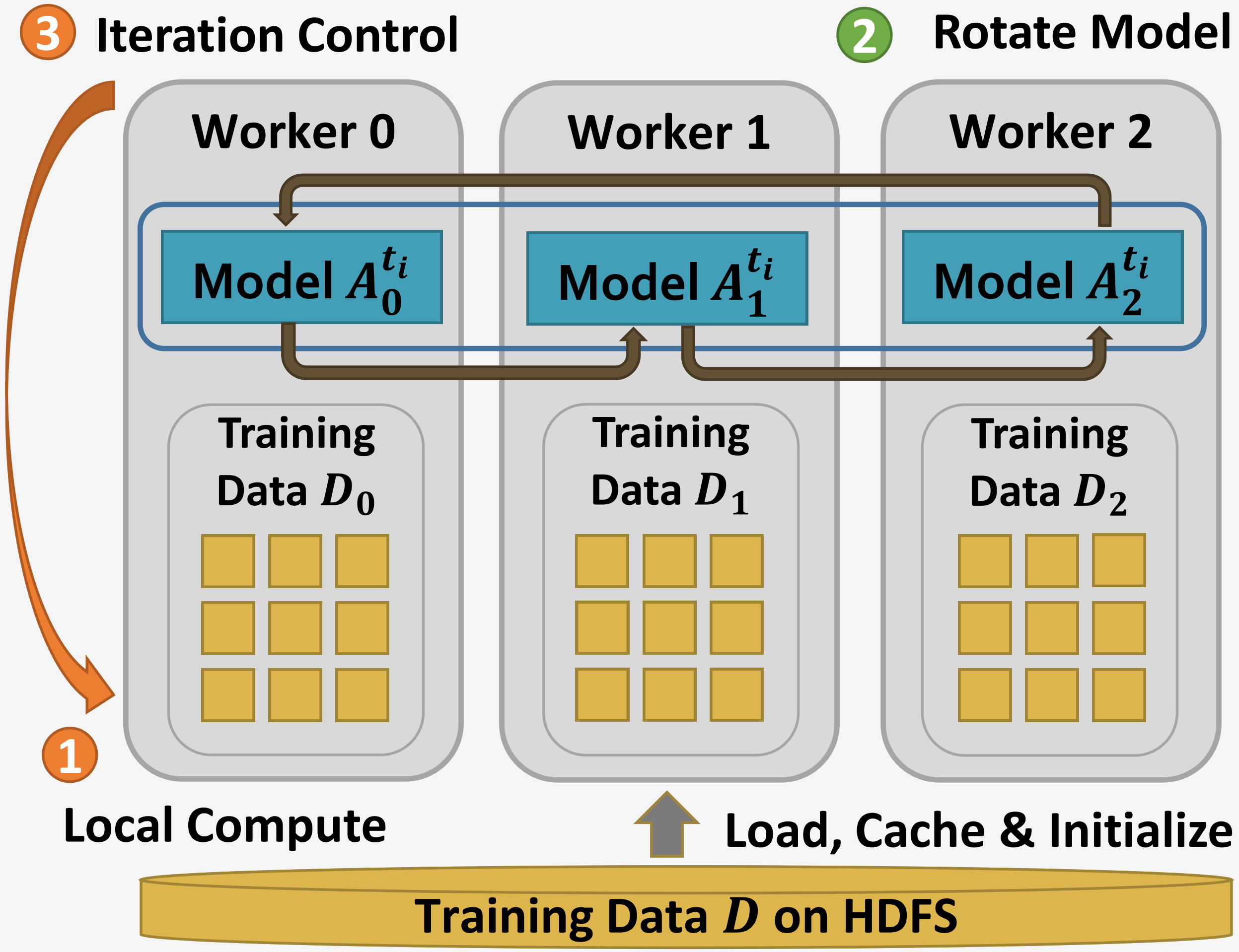
Algorithm Examples

Collapsed Gibbs Sampling for Latent Dirichlet Allocation

Stochastic Gradient Descent for Matrix Factorization

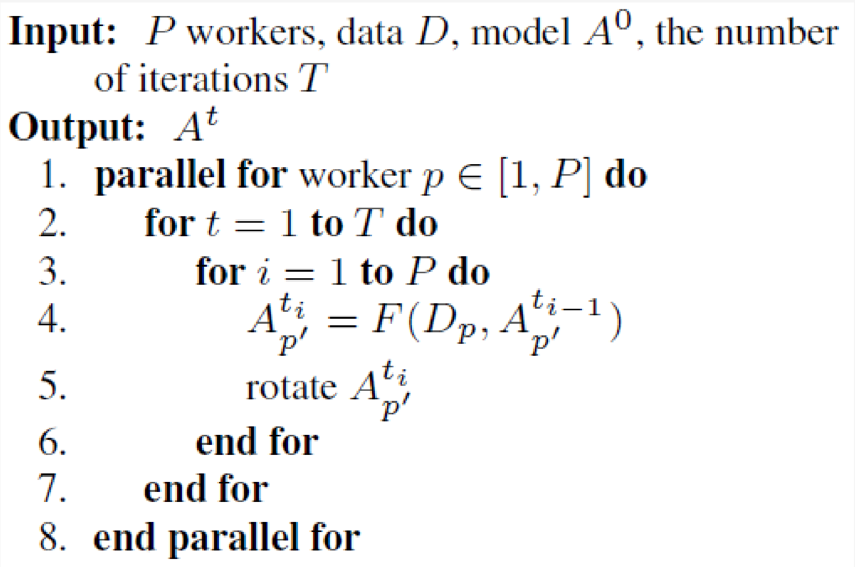
Cyclic Coordinate Descent for Matrix Factorization

A Parallelization Solution using Model Rotation



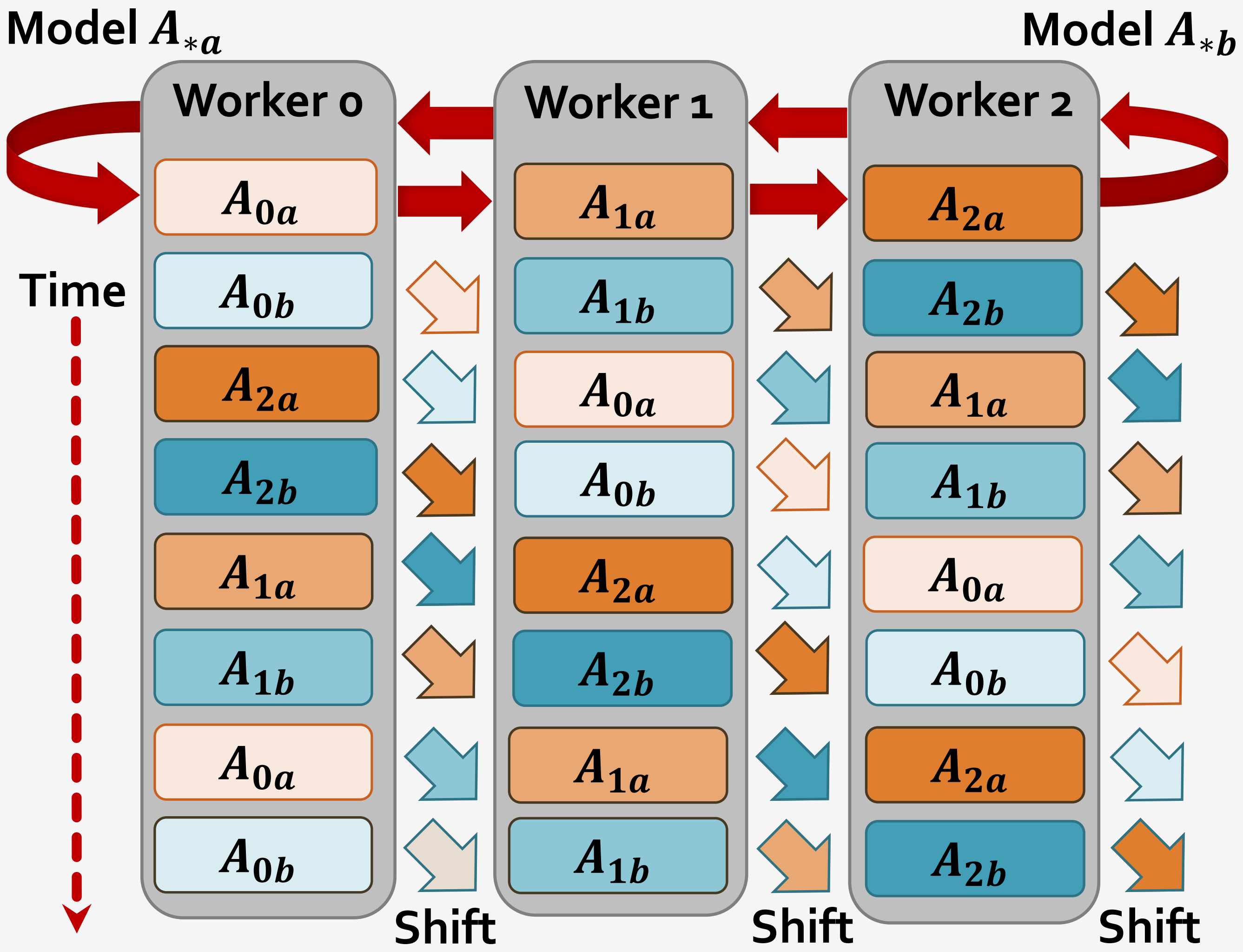
Maximizing the effectiveness of parallel model updates for algorithm convergence

Minimizing the overhead of communication for scaling



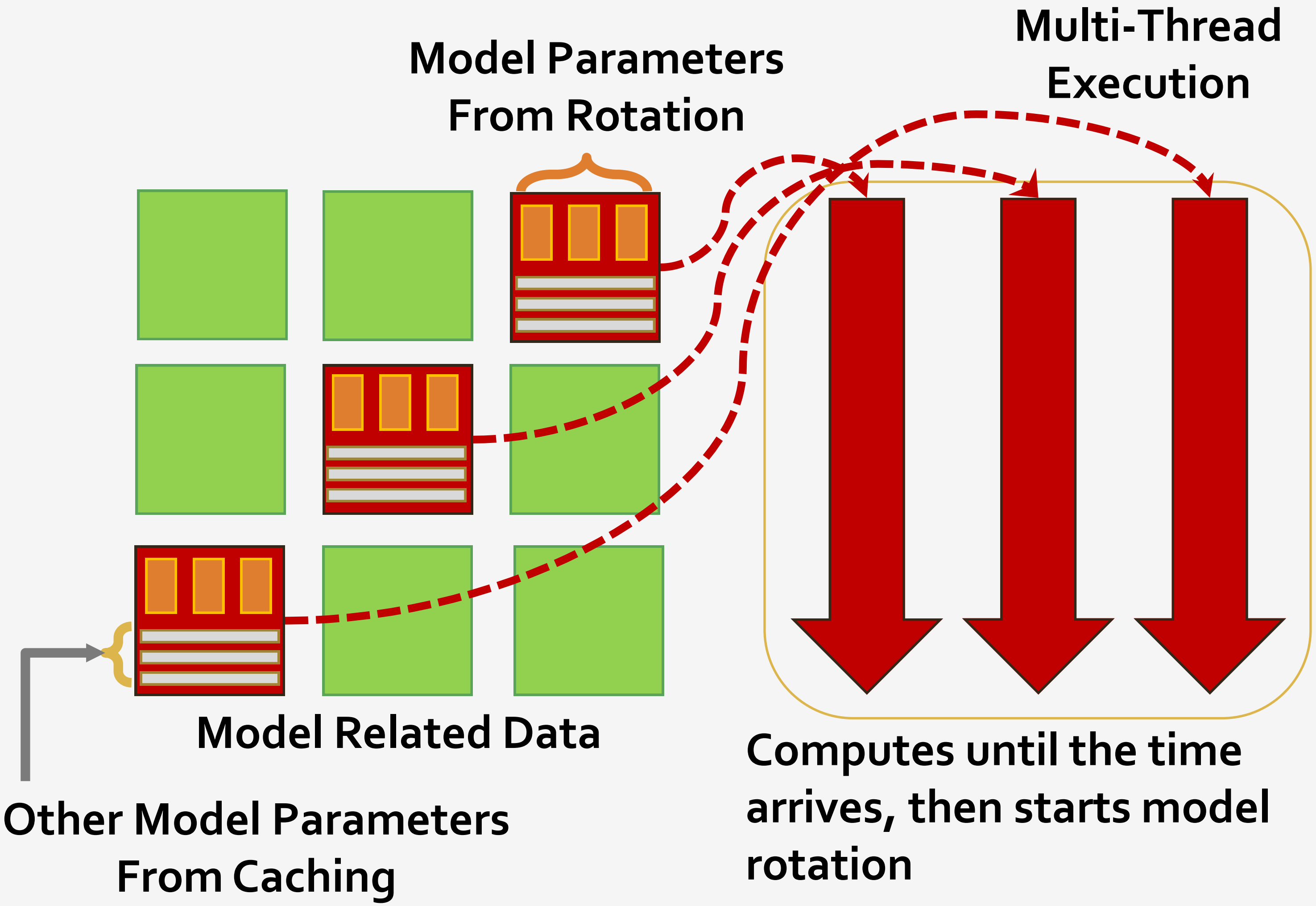
```
Input: $P$ workers, data $D$, model $A^0$, the number
       of iterations $T$
Output: $A^t$
parallel for worker $p \in [1, P]$ do
    for $t = 1$ to $T$ do
        for $i = 1$ to $P$ do
            $A_{p'}^{t_i} = F(D_p, A_{p'}^{t_{i-1}})$
            rotate $A_{p'}^{t_i}$
        end for
    end for
end parallel for
```

Pipeline Model Rotation



Worker 0	Worker 1	Worker 2
A_{0a}	A_{1a}	A_{2a}
A_{0b}	A_{1b}	A_{2b}
A_{2a}	A_{0a}	A_{1a}
A_{2b}	A_{0b}	A_{1b}
A_{1a}	A_{2a}	A_{0a}
A_{1b}	A_{2b}	A_{0b}
A_{0a}	A_{1a}	A_{2a}
A_{0b}	A_{1b}	A_{2b}

Dynamic Rotation Control for LDA CGS and MF SGD



Model Parameters From Rotation

Multi-Thread Execution

Model Related Data

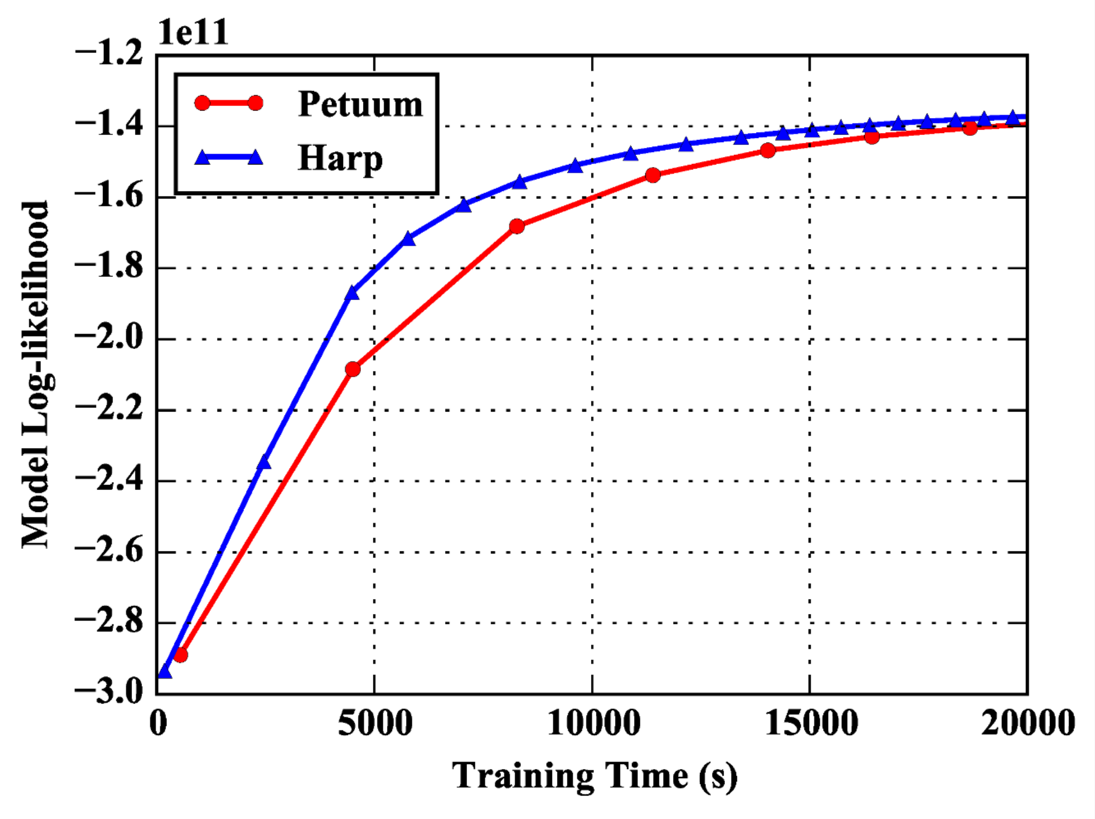
Other Model Parameters From Caching

Computes until the time arrives, then starts model rotation

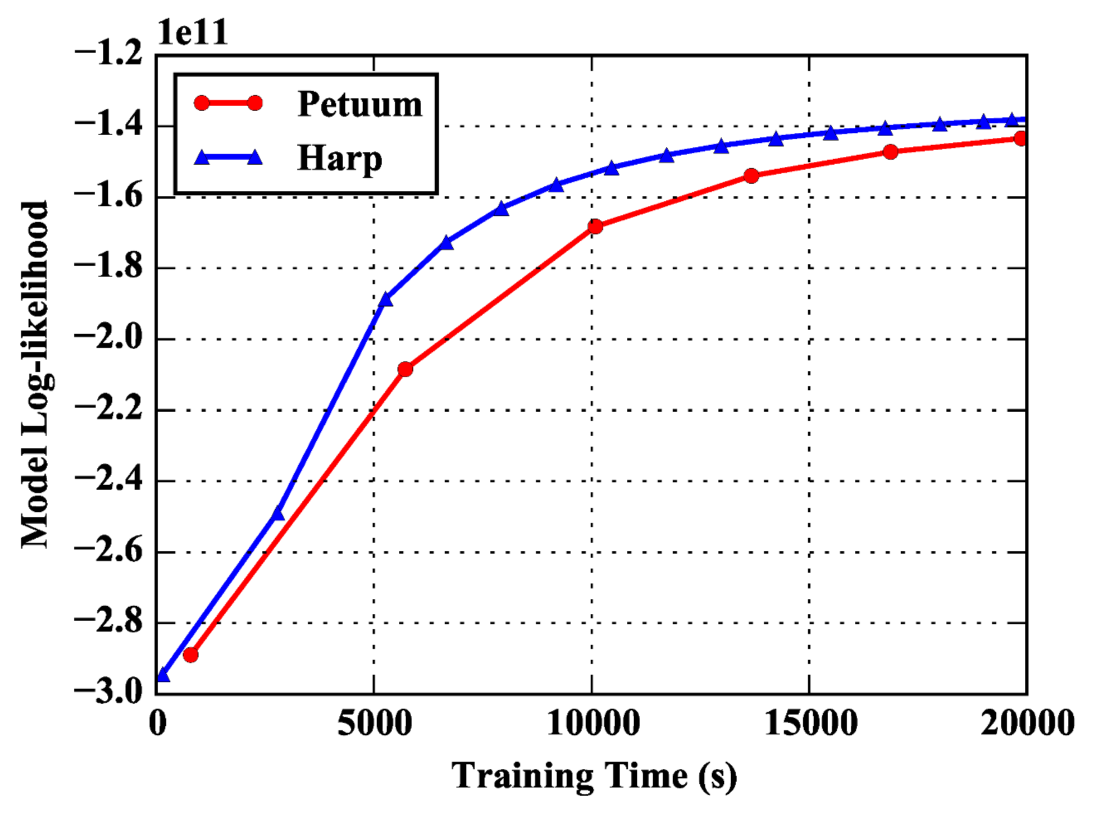
SALSA

CGS Model Convergence Speed

LDA Dataset	Documents	Words	Tokens	CGS Parameters
clueweb1	76163963	999933	29911407874	$K = 10000, \alpha = 0.01, \beta = 0.01$



60 nodes x 20 threads/node



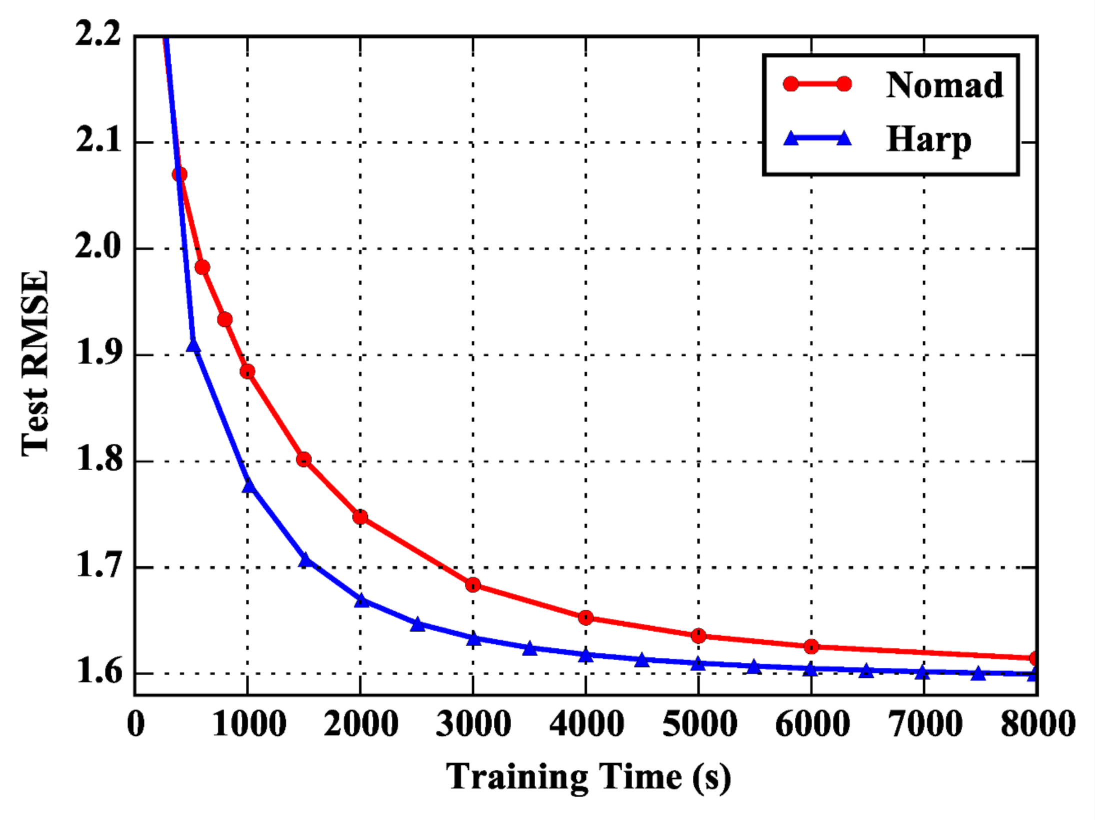
30 nodes x 30 threads/node

K: number of features; *a*, *b* hyperparameters;

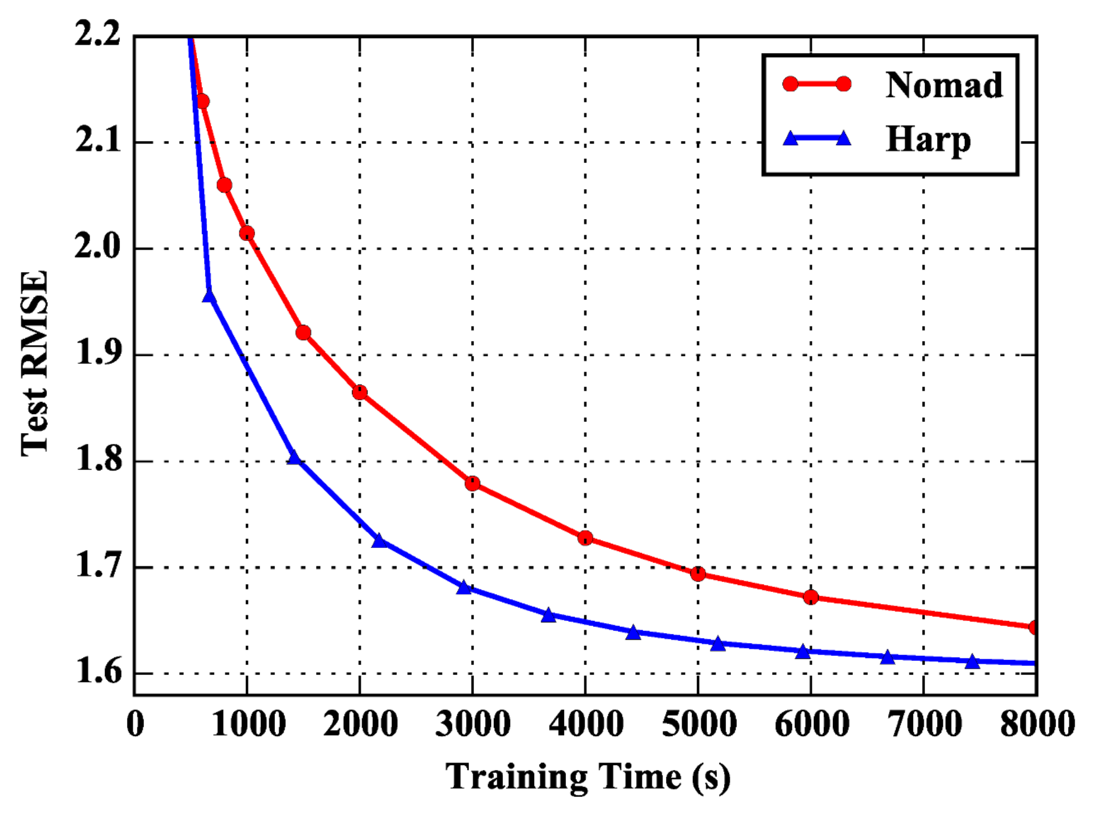
SALSA

SGD Model Convergence Speed

MF Dataset	Rows	Columns	Non-ZeroElements	SGD Parameters
clueweb2	76163963	999933	15997649665	$K = 2000, \lambda = 0.01, \epsilon = 0.001$



60 nodes x 20 threads/node

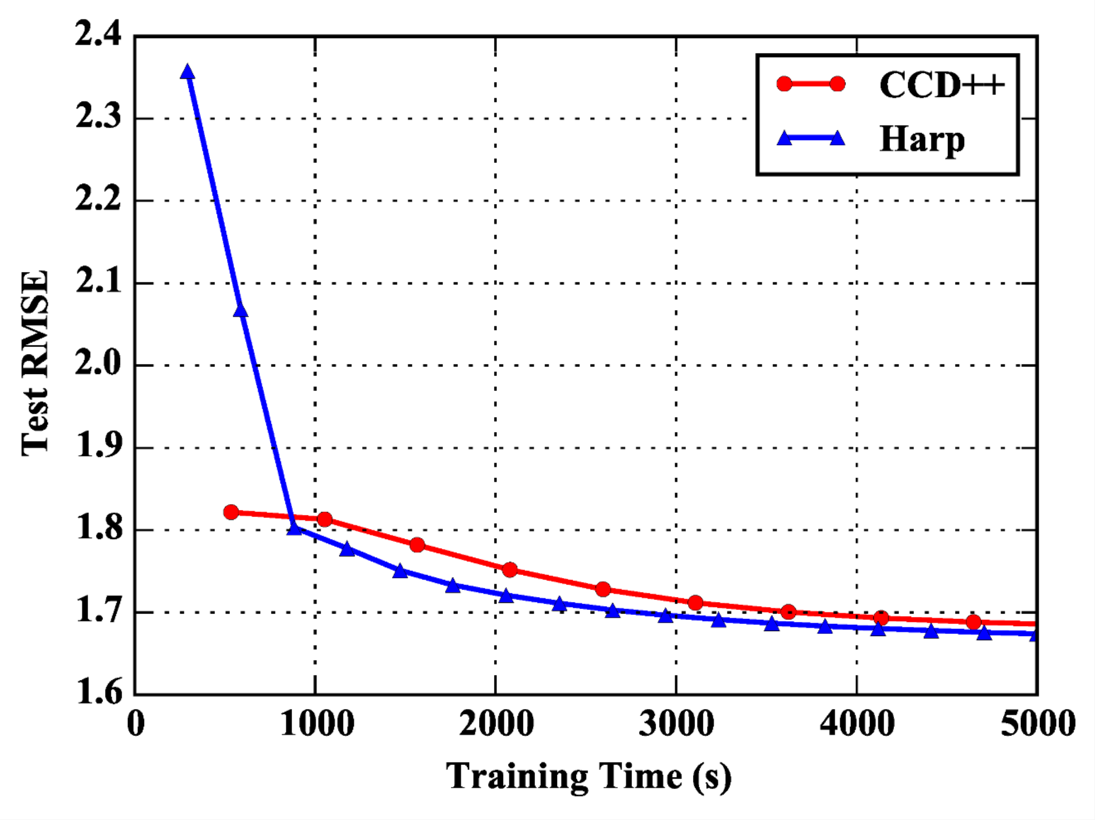


30 nodes x 30 threads/node

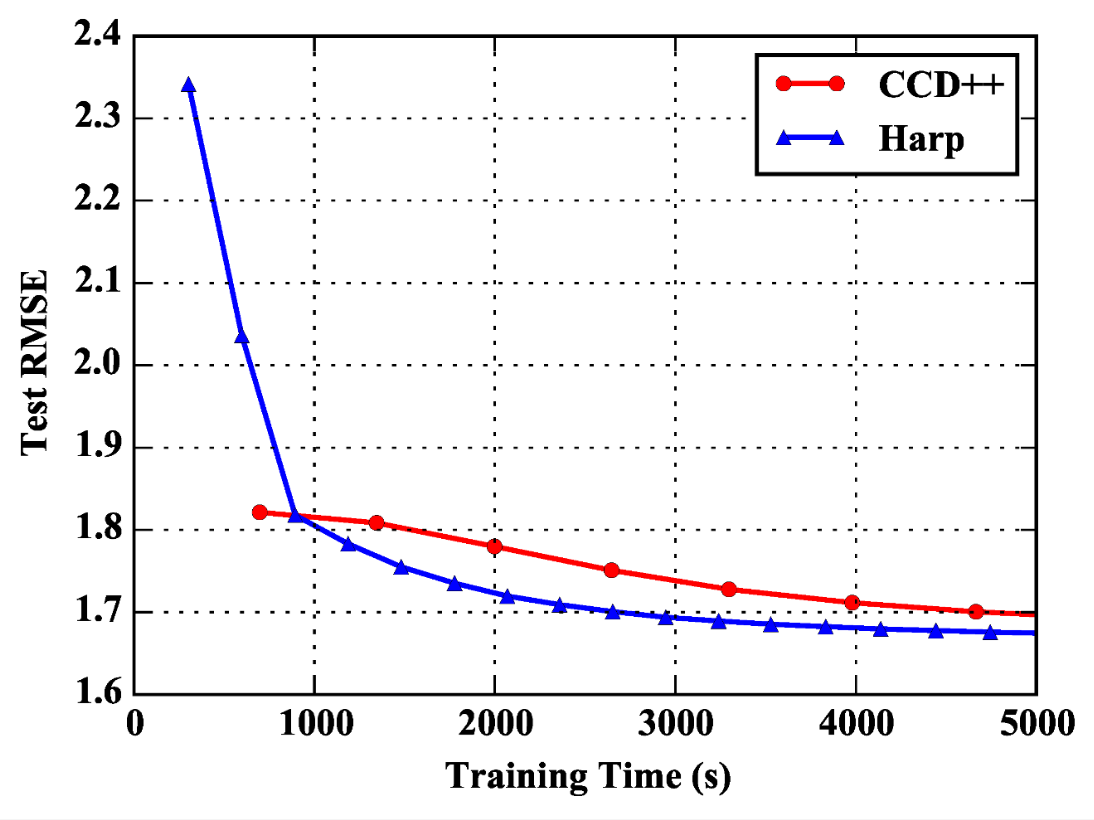
K: number of features; λ regularization parameter; ϵ learning rate

CCD Model Convergence Speed

MF Dataset	Rows	Columns	Non-Zero Elements	CCD Parameters
clueweb2	76163963	999933	15997649665	$K = 120, \quad \lambda = 0.1$



60 nodes x 20 threads/node



30 nodes x 30 threads/node

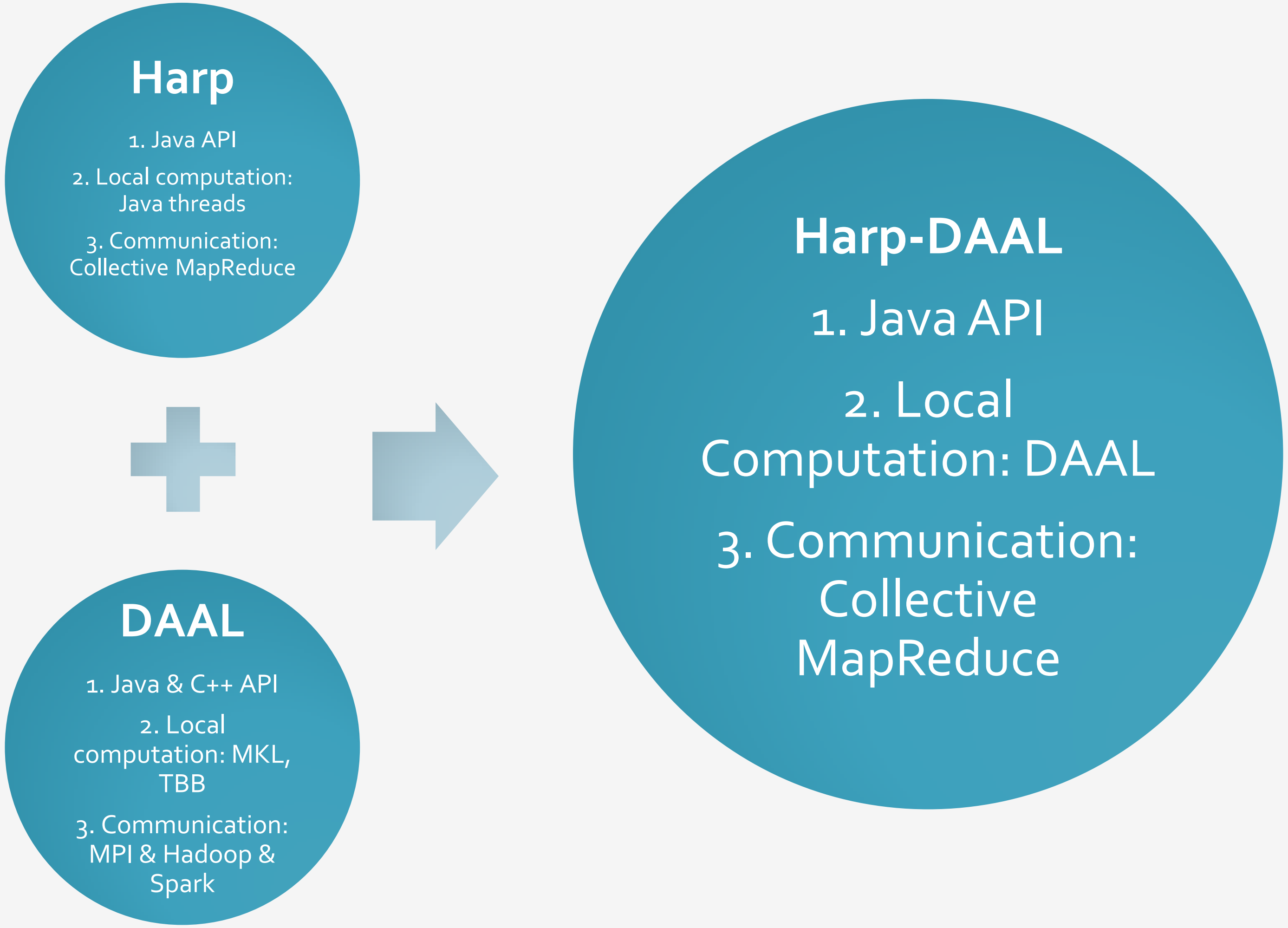
K: number of features; λ regularization parameter

SALSA

Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work

Harp-DAAL: High Performance Machine Learning Framework



Harp

1. Java API
2. Local computation: Java threads
3. Communication: Collective MapReduce

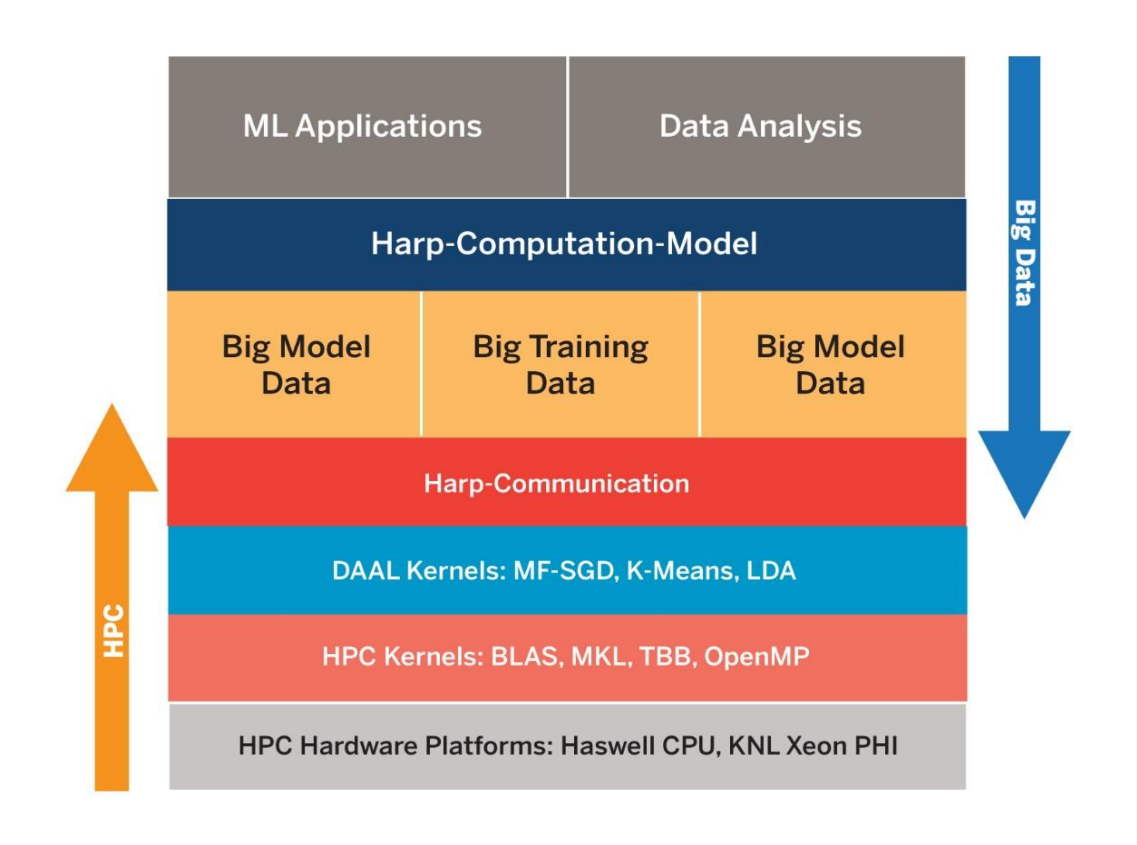
DAAL

1. Java & C++ API
2. Local computation: MKL, TBB
3. Communication: MPI & Hadoop & Spark

Harp-DAAL

1. Java API
2. Local Computation: DAAL
3. Communication: Collective MapReduce

Harp-DAAL in the HPC-BigData Stack



ML Applications

Data Analysis

Harp-Computation-Model

Big Model Data

Big Training Data

Big Model Data

Harp-Communication

DAAL Kernels: MF-SGD, K-Means, LDA

HPC Kernels: BLAS, MKL, TBB, OpenMP

HPC Hardware Platforms: Haswell CPU, KNL Xeon PHI

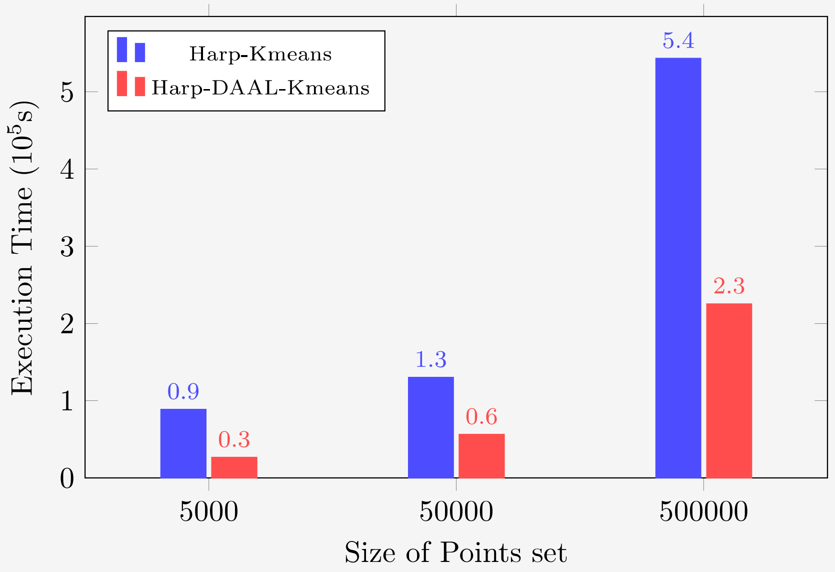
Harp-DAAL is at the intersection of HPC and Big Data stacks, which requires:

- Interface: User friendly, consistent with other Java written Data analytics Apps.
- Low level Kernels: highly optimized for HPC platforms such as many-core architecture
- Models: inherit Harp's computation models for different ML algorithms

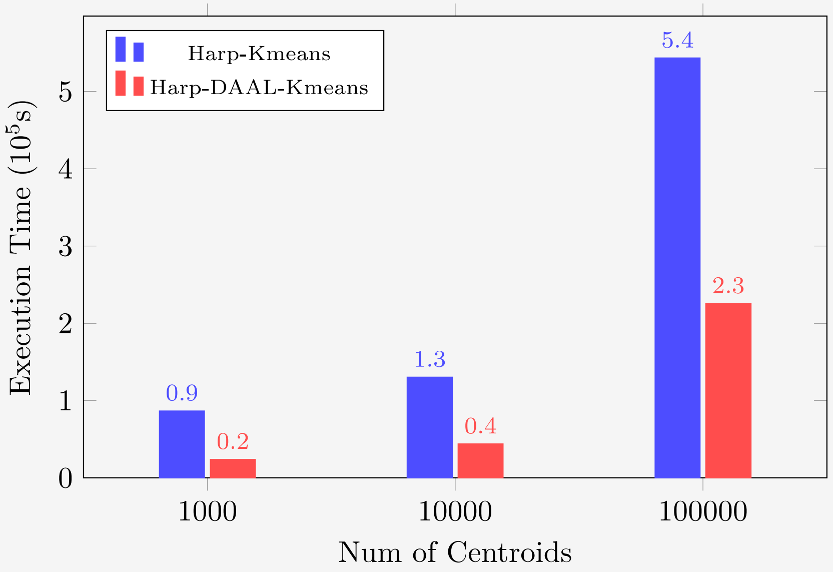
Inter-node Performance: Harp-DAAL-Kmeans vs. Harp-Kmeans

Inter-node test is done on two Haswell E5-2670 v3 2.3GHz nodes. We vary the size of input points and the number of centroids (clusters)

By using DAAL-Harp's high performance kernels, DAAL-Harp-Kmeans has a 2x to 4x speeds up over Harp-Kmeans



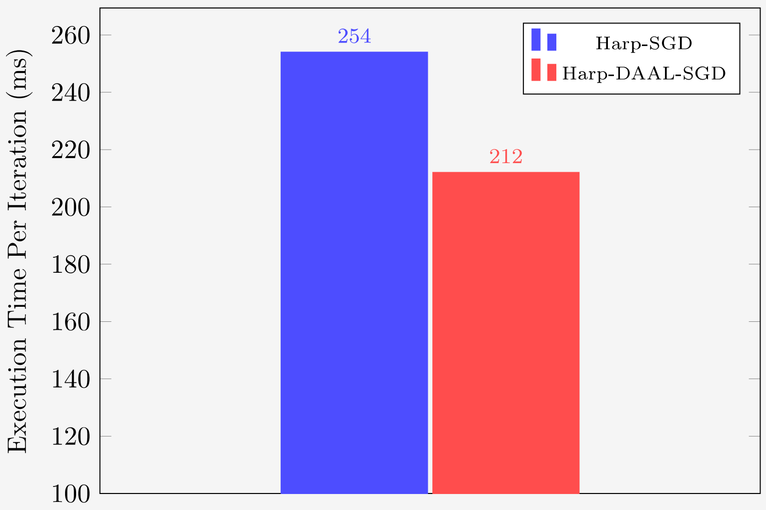
Harp-Kmeans vs. Harp-DAAL-Kmeans with 100000 Centroids



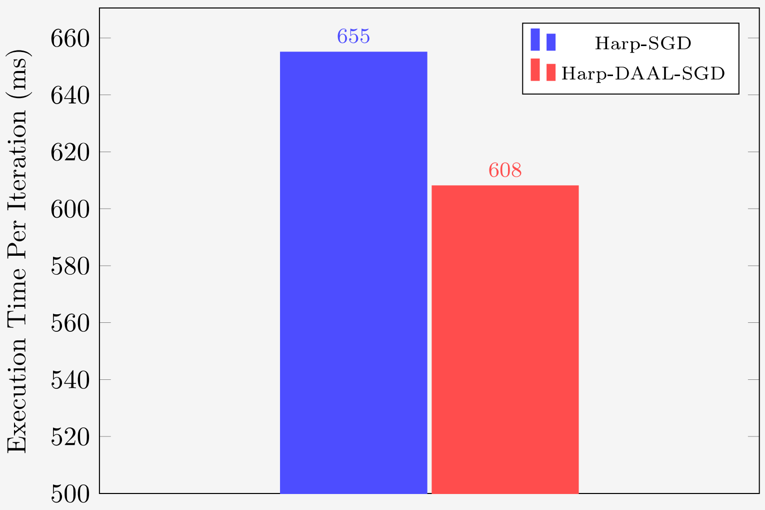
Harp-Kmeans vs. Harp-DAAL-Kmeans with 500000 Points

Inter-node Performance: Harp-DAAL-SGD vs. Harp-SGD

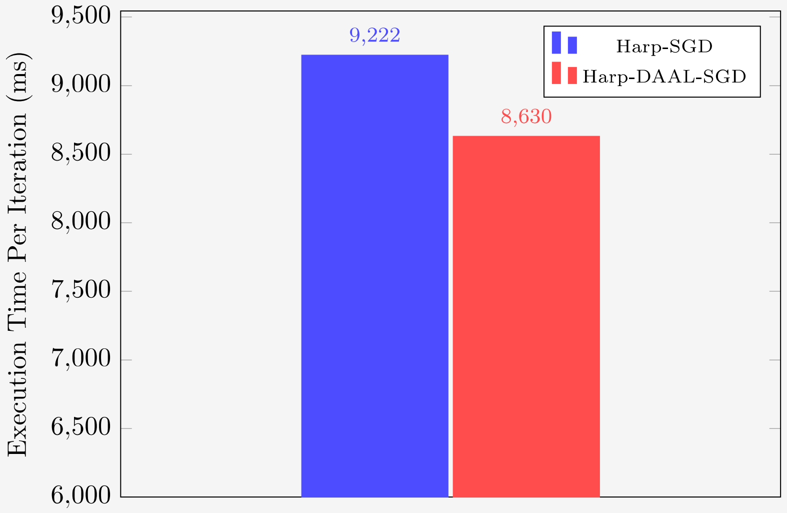
Test for MovieLens with Dim 128 on 2 nodes of Juliet



Test for MovieLens with Dim 512 on 2 nodes of Juliet



Test for Yahoomusic with Dim 128 on 2 nodes of Juliet



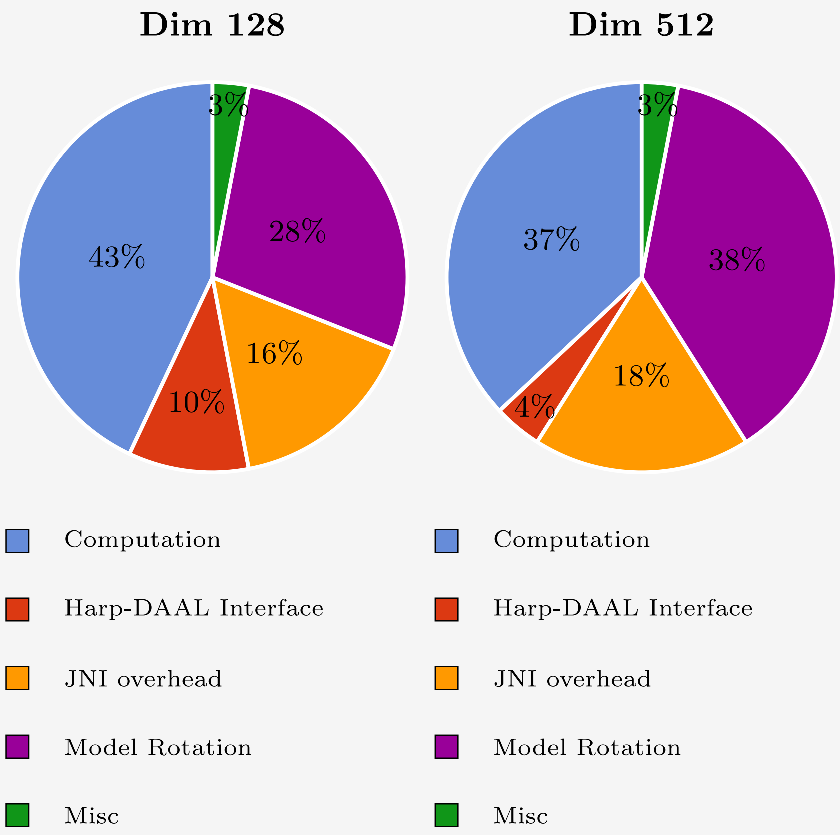
The Inter-node test is done on two Haswell E5-2670 v3 2.3GHz nodes. We use two datasets

- MovieLens, a small set with 9301274 points
- Yahoomusic a large set with 252800275 points

For both datasets, we have around **5% to 15% speeds up** by using DAAL-SGD within Harp.

There are still some overheads of interfacing DAAL and Harp, which requires further investigation.

Interface Overhead between DAAL and Harp



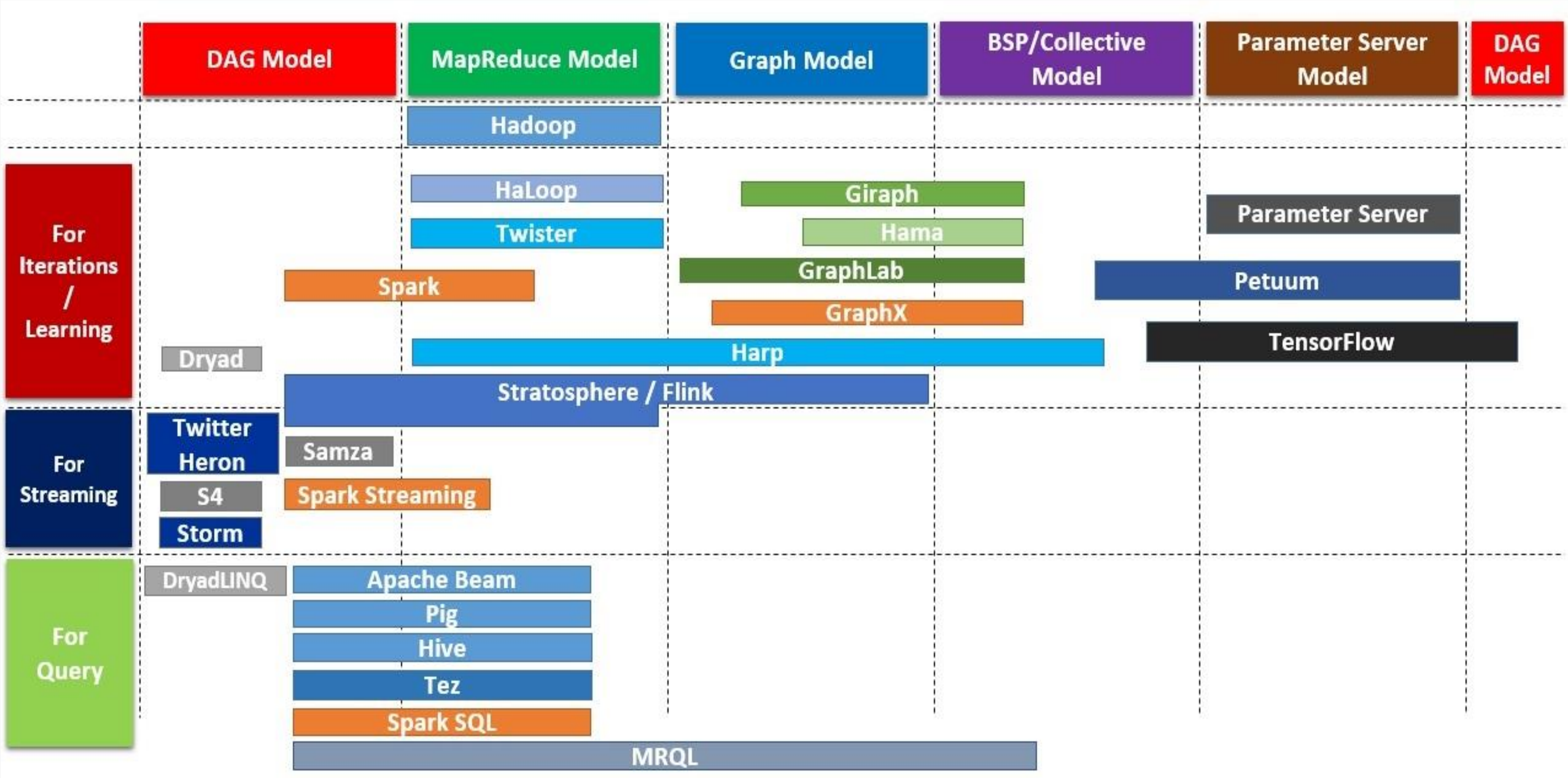
We decompose the training time into different phases. There are two overhead of interface

- Harp-DAAL Interface
  - Conversion between data structures
- JNI interface
  - Data movement from Java heap to out-of-heap buffer for C++ native kernels in DAAL
- The two overheads could take up to 25% of the total training time, which must be optimized in the future work.
- Rewrite some Harp codes to create shared memory space between DAAL and Harp
- Add more Harp compatible data structures to DAAL

Outline

1. Motivation: Machine Learning Applications
2. Harp-DAAL Framework: Design and Implementations
3. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
4. Conclusions and Future Work

The Models of Contemporary Big Data Tools



DAG Model

MapReduce Model

Graph Model

BSP/Collective Model

Parameter Server Model

DAG Model

Hadoop

For Iterations / Learning

HaLoop

Twister

Giraph

Hama

Parameter Server

Spark

GraphLab

Petuum

GraphX

TensorFlow

Dryad

Harp

Stratosphere / Flink

For Streaming

Twitter Heron

Samza

S4

Spark Streaming

Storm

For Query

DryadLINQ

Apache Beam

Pig

Hive

Tez

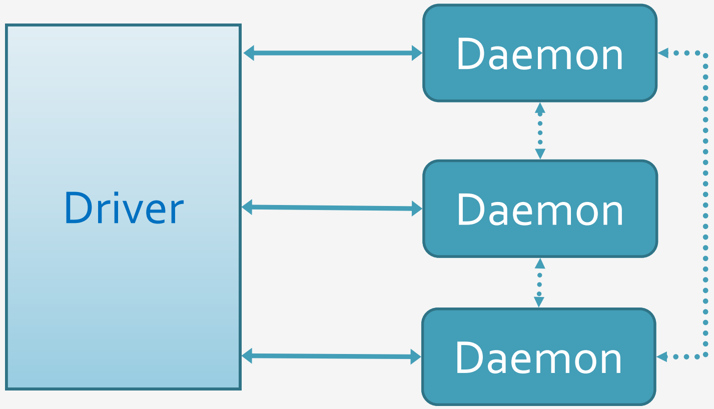
Spark SQL

MRQL

Programming Models

Comparison of Iterative Computation Tools

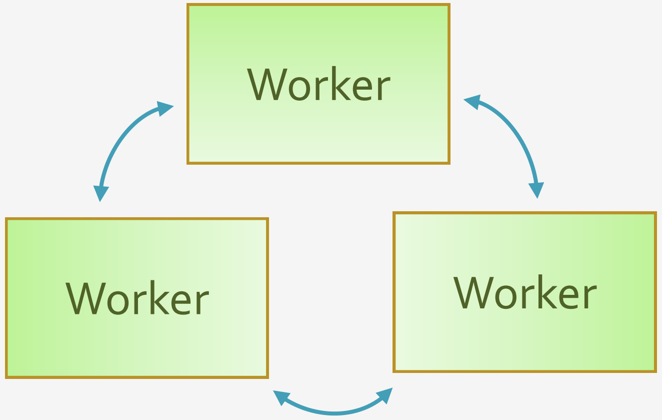
Spark



- Implicit Data Partitioning
- Implicit Communication

M. Zaharia et al. "Spark: Cluster Computing with Working Sets". HotCloud, 2010.

Harp

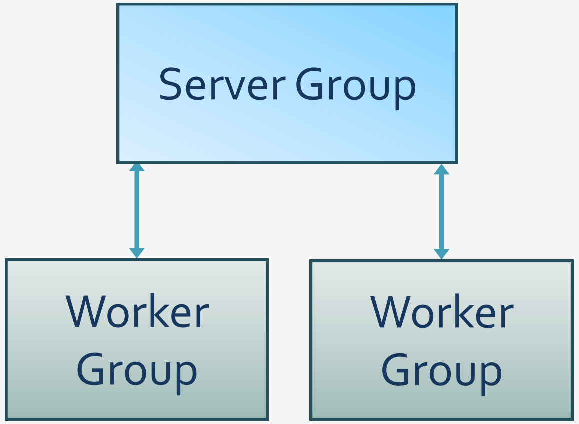


Various Collective Communication Operations

- Explicit Data Partitioning
- Explicit Communication

B. Zhang, Y. Ruan, J. Qiu. "Harp: Collective Communication on Hadoop". IC2E, 2015.

Parameter Server



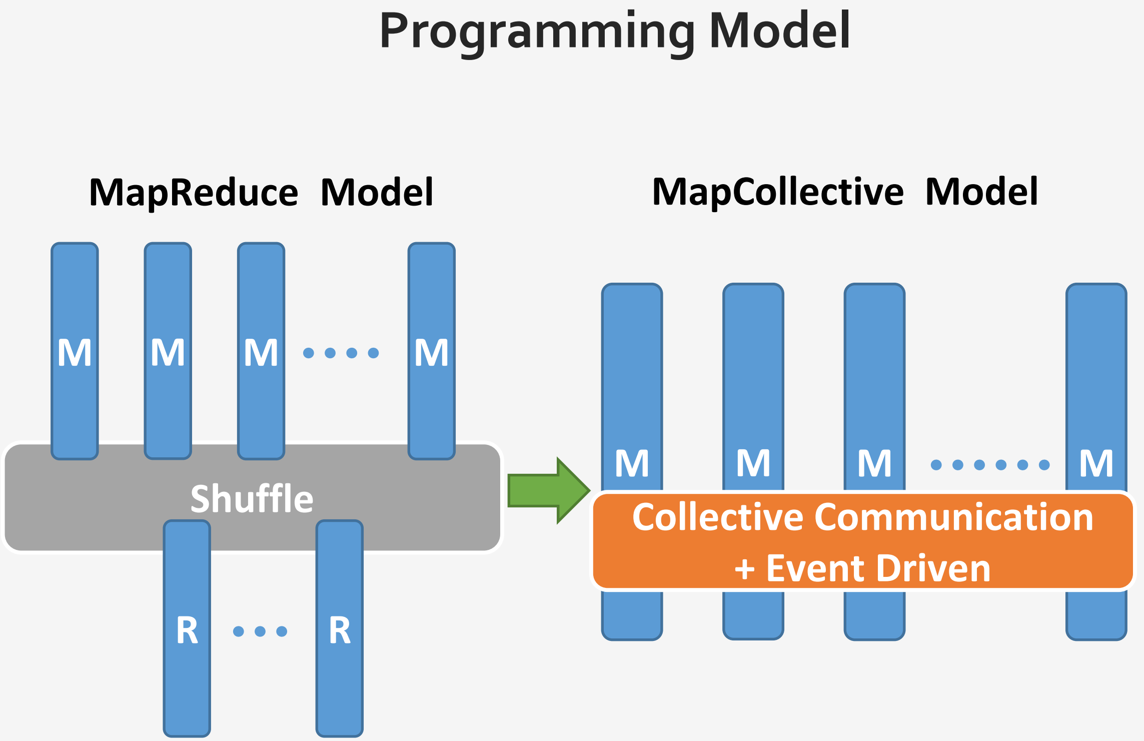
Asynchronous Communication Operations

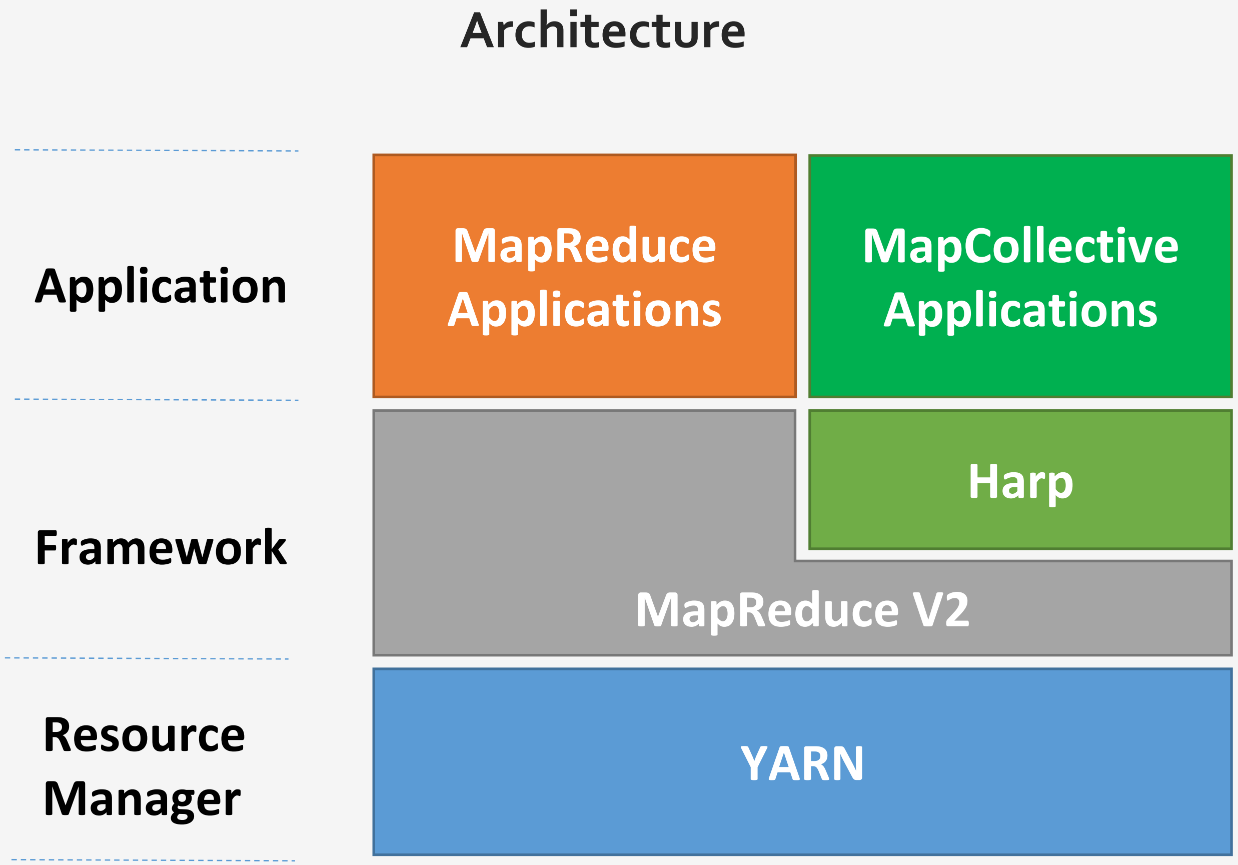
- Explicit Data Partitioning
- Implicit Communication

M. Li, et al. "Scaling Distributed Machine Learning with the Parameter Server". OSDI, 2014.

SALSA

Harp: a Hadoop plug-in based on map-collective models

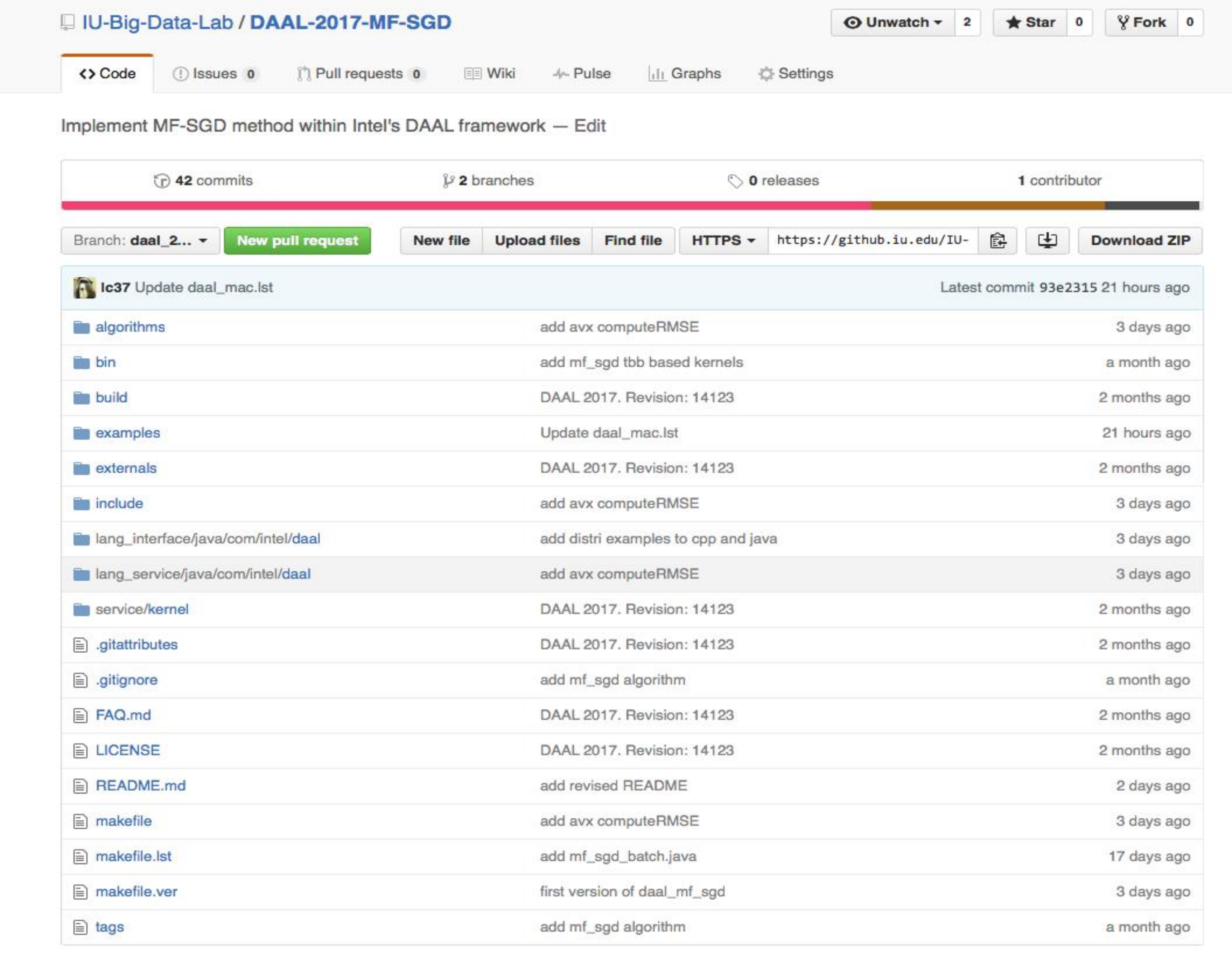




- MPI-like collective communication operations that are highly optimized for big data problems.
- A Hadoop Plug-in to integrate with the ecosystems.
- Efficient and innovative computation models for different machine learning problems.

SALSA

Hadoop / Harp-DAAL: Prototype and Production Code



IU-Big-Data-Lab / **DAAL-2017-MF-SGD**

Implement MF-SGD method within Intel's DAAL framework — Edit

lc37 Update daal_mac.lst — Latest commit 93e2315 21 hours ago

algorithms	add avx computeRMSE	3 days ago
bin	add mf_sgd tbb based kernels	a month ago
build	DAAL 2017. Revision: 14123	2 months ago
examples	Update daal_mac.lst	21 hours ago
externals	DAAL 2017. Revision: 14123	2 months ago
include	add avx computeRMSE	3 days ago
lang_interface/java/com/intel/daal	add distri examples to cpp and java	3 days ago
lang_service/java/com/intel/daal	add avx computeRMSE	3 days ago
service/kernel	DAAL 2017. Revision: 14123	2 months ago
.gitattributes	DAAL 2017. Revision: 14123	2 months ago
.gitignore	add mf_sgd algorithm	a month ago
FAQ.md	DAAL 2017. Revision: 14123	2 months ago
LICENSE	DAAL 2017. Revision: 14123	2 months ago
README.md	add revised README	2 days ago
makefile	add avx computeRMSE	3 days ago
makefile.lst	add mf_sgd_batch.java	17 days ago
makefile.ver	first version of daal_mf_sgd	3 days ago
tags	add mf_sgd algorithm	a month ago

Source codes available on Indiana University's Github account

An example of MF-SGD is at

https://github.iu.edu/IU-Big-Data-Lab/DAAL-2017-MF-SGD

- Harp-DAAL follows the same standard of DAAL's original codes
- improve DAAL's existed algorithms for distributed usage
- add new algorithms to DAAL's codebase.
- Harp-DAAL's kernel is also compatible with other communication tools.

Summary and Future Work

- Identification of **Apache Big Data Software Stack** and integration with **High Performance Computing Stack** to give **HPC-ABDS**
  - ABDS (Many Big Data applications/algorithms need HPC for performance)
  - HPC (needs software model productivity/sustainability)
- Identification of **4 computation models** for machine learning applications
- **HPC-ABDS Plugin Harp:** adds HPC communication performance and rich data abstractions to Hadoop by development of Harp library of **Collectives** to use at Reduce phase
  - Broadcast and Gather needed by current applications
  - Discover other important ones (e.g. Allgather, Global-local sync, Rotation pipeline)
- Integration of **Hadoop/Harp** with Intel **DAAL** and other libraries
- Implement efficiently on each platform (e.g. Amazon, Azure, Big Red II, Haswell/KNL Clusters)