Benchmarking Harp-DAAL: High Performance Hadoop on KNL Clusters

IEEE Cloud Computing Conference

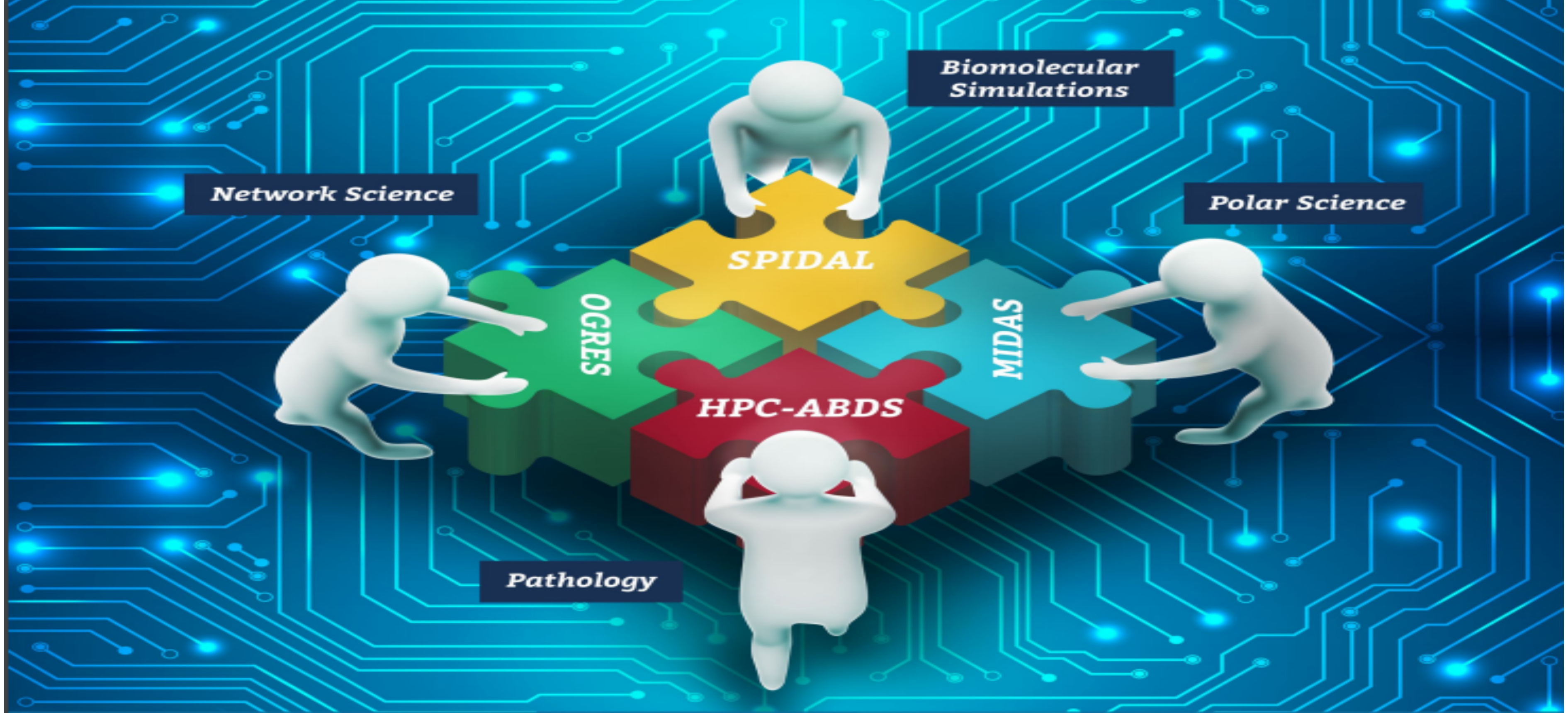
June 27, 2017

Judy Qiu

Intelligent Systems Engineering Department, Indiana University
Email: xqiu@indiana.edu

Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work



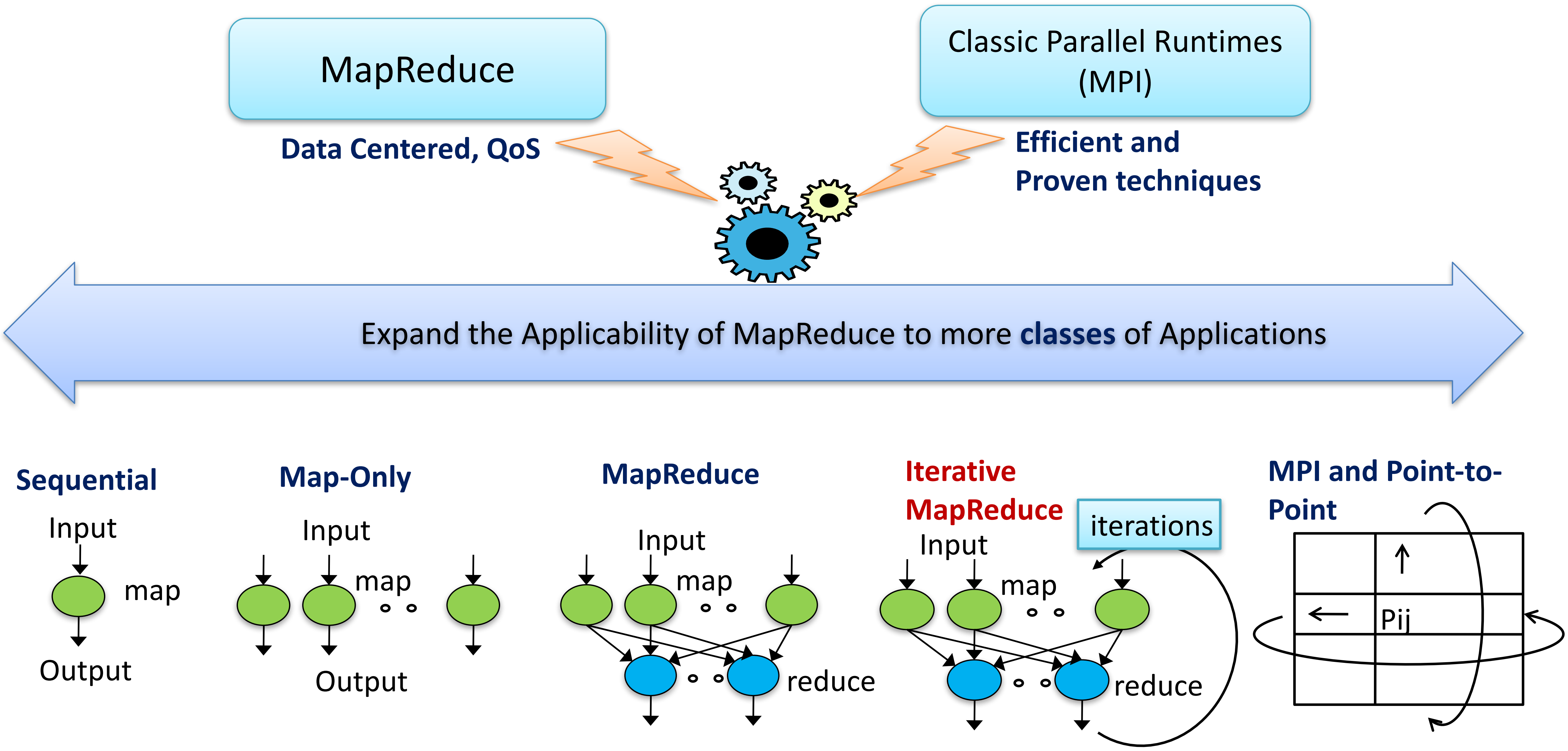
DATANET: CIF21 DIBBS: Middleware and High Performance Analytics Libraries for Scalable Data Science Progress Report August 2016

Indiana University (Fox, Qiu, Crandall, von Laszewski), Rutgers (Jha), Virginia Tech (Marathe), Kansas (Paden), Stony Brook (Wang), Arizona State (Beckstein), Utah (Cheatham)

Motivation for faster and bigger problems

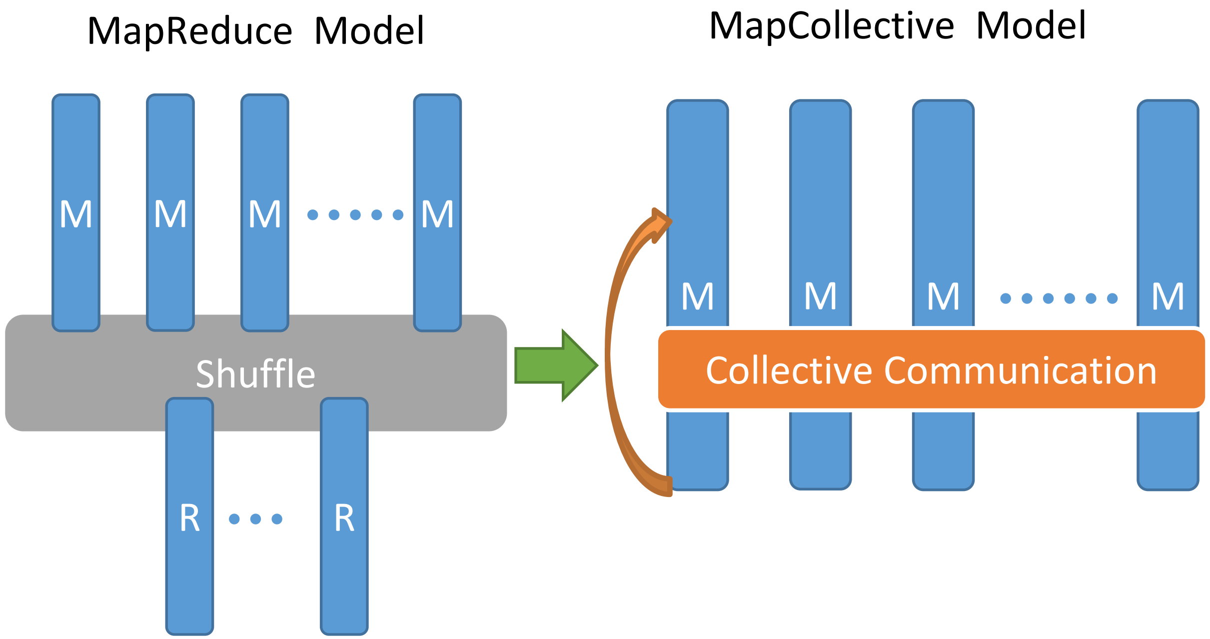
- **Machine Learning** (**ML**) Needs high performance
  - **Big data** and **Big model**
  - **Iterative algorithms** are fundamental in learning a non-trivial model
  - **Model training** and **Hyper-parameter tuning** steps run the iterative algorithms many times
- Architecture for Big Data analytics
  - to understand the algorithms through a **Model-Centric** view
  - to focus on the **computation and communication patterns** for optimizations
  - Trade-offs of efficiency and productivity
    - linear speedup with an increasing number of processors
    - easier to be parallelized on multicore or manycore computers

High Performance – Apache Big Data Stack

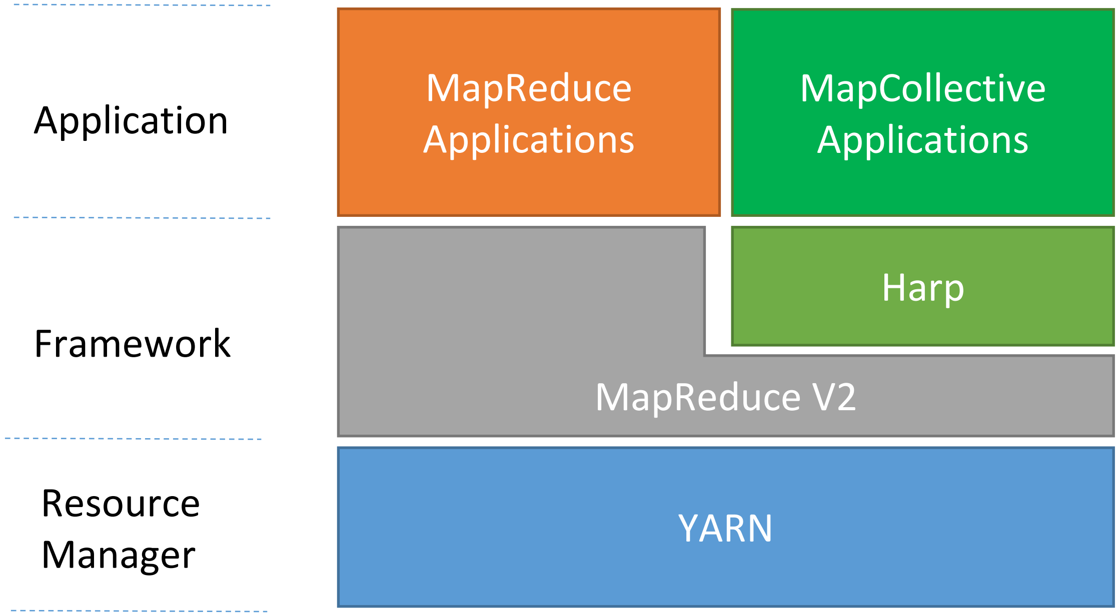


The Concept of Harp Plug-in

Parallelism Model



Architecture

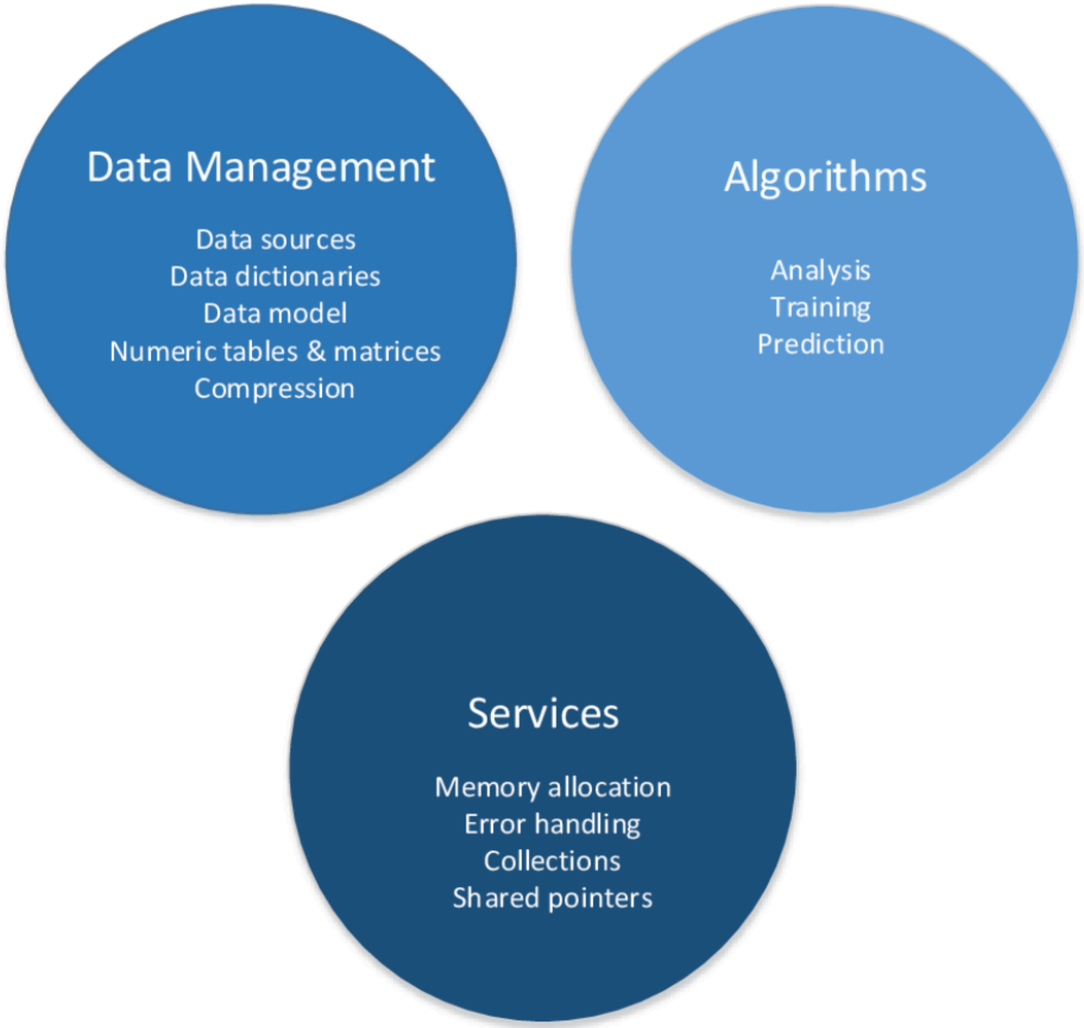


Harp is an open-source project developed at Indiana University [6], it has:

- 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.

[6] Harp project. Available at https://dsc-spidal.github.io/harp

Developer Zone DAAL: Intel® Data Analytics Acceleration Library

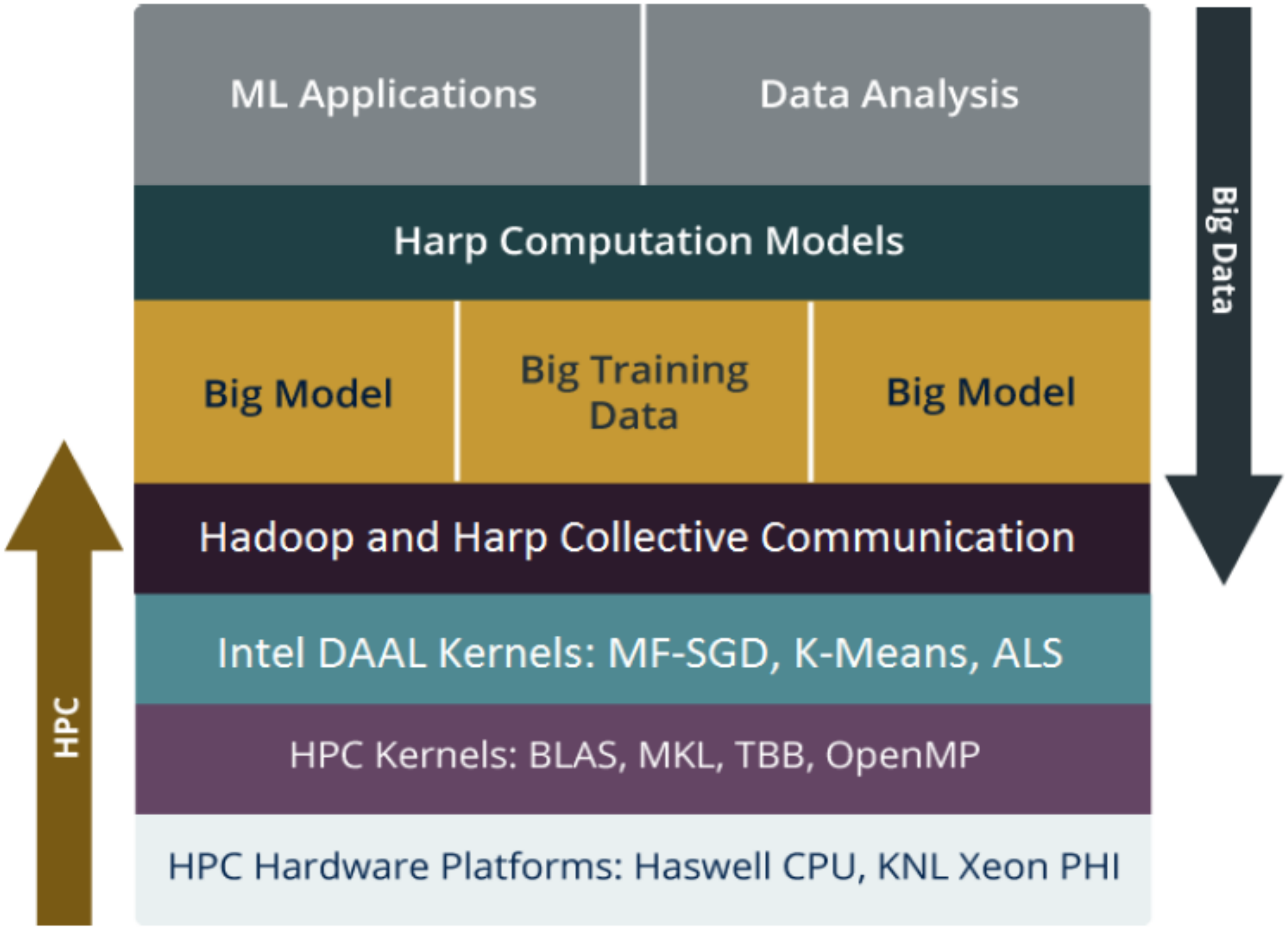


DAAL is an open-source project that provides:

- 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

Harp-DAAL enable faster Machine Learning Algorithms

with Hadoop Clusters on Multi-core and Many-core architectures

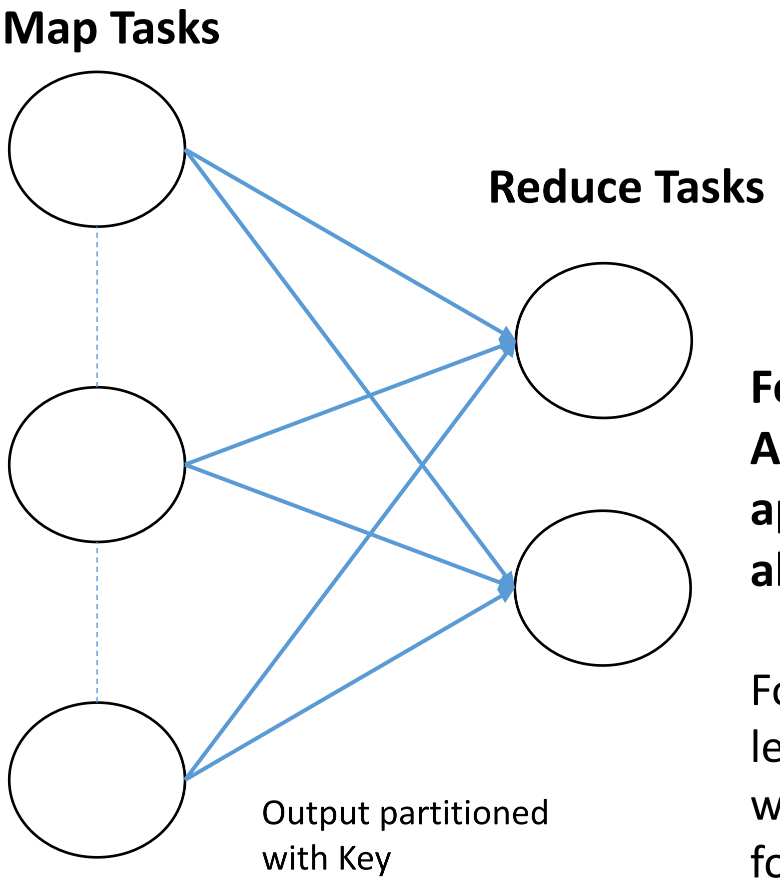


- Bridge the gap between HPC hardware and Big data/Machine learning Software
- Support Iterative Computation, Collective Communication, Intel DAAL and native kernels
- Portable to new many-core architectures like Xeon Phi and run on Haswell and KNL clusters

Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work

General Reduction in Hadoop, Spark, Flink



Follow by Broadcast for AllReduce which is a common approach to support iterative algorithms

For example, paper [7] 10 learning algorithms can be written in a certain “summation form,” which allows them to be easily parallelized on multicore computers.

Comparison of Reductions:

- Separate Map and Reduce Tasks
- Switching tasks is expensive
- MPI only has one sets of tasks for map and reduce
- MPI achieves AllReduce by interleaving multiple binary trees
- MPI gets efficiency by using shared memory intra-node (e.g. multi-/manycore, GPU)

[7] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng and Kunle Olukotun, Map-Reduce for Machine Learning on Multicore, in NIPS 19 2007.

HPC Runtime versus ABDS distributed Computing Model on Data Analytics

Hadoop writes to disk and is slowest; Spark and Flink spawn many processes and do not support allreduce directly; MPI does in-place combined reduce/broadcast

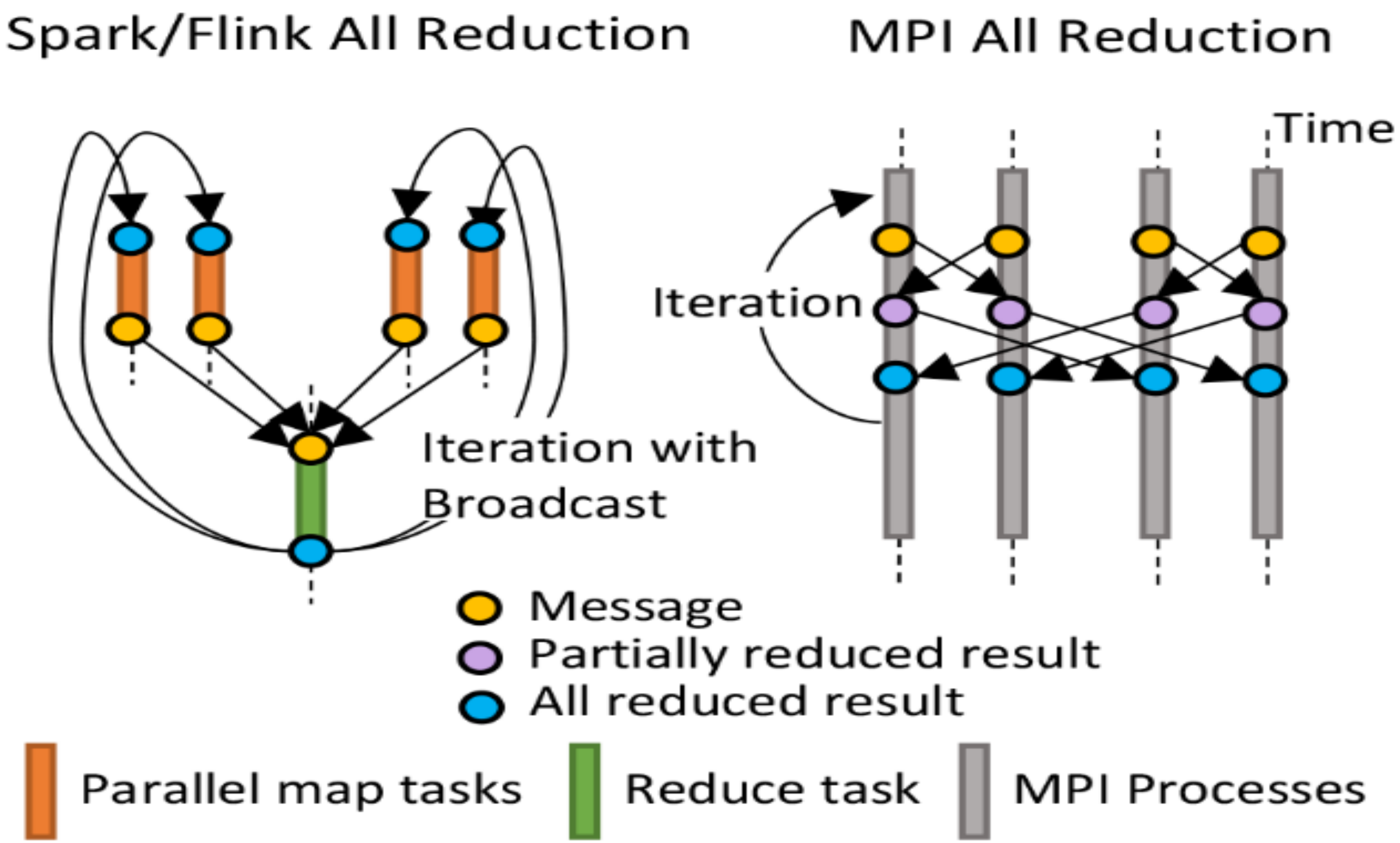
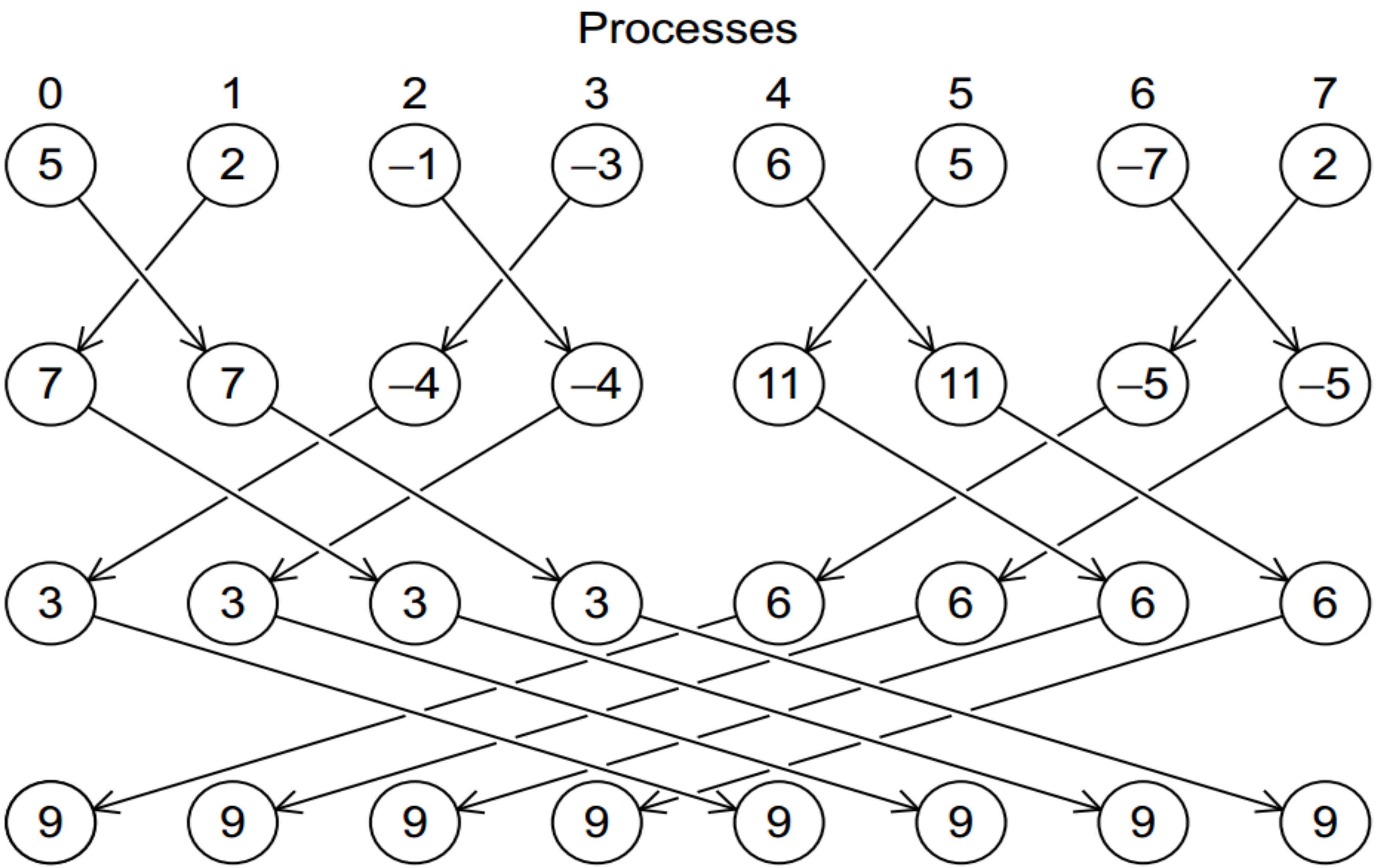


Illustration of In-Place AllReduce in MPI



Why Collective Communications for Big Data Processing?

- **Collective Communication and Data Abstractions**
  - Optimization of global model synchronization
    - ML algorithms: convergence vs. consistency
    - Model updates can be out of order
  - Hierarchical data abstractions and operations
- **Map-Collective Programming Model**
  - Extended from MapReduce model to support collective communications
  - BSP parallelism at Inter-node vs. Intra-node levels
- **Harp Implementation**
  - A plug-in to Hadoop

Harp APIs

Scheduler

- DynamicScheduler
- StaticScheduler

Collective

- **MPI collective communication**
  - broadcast
  - reduce
  - allgather
  - allreduce
- **MapReduce “shuffle-reduce”**
  - regroup with combine
- **Graph & ML operations**
  - “push” & “pull” model parameters
  - rotate global model parameters between workers

Event Driven

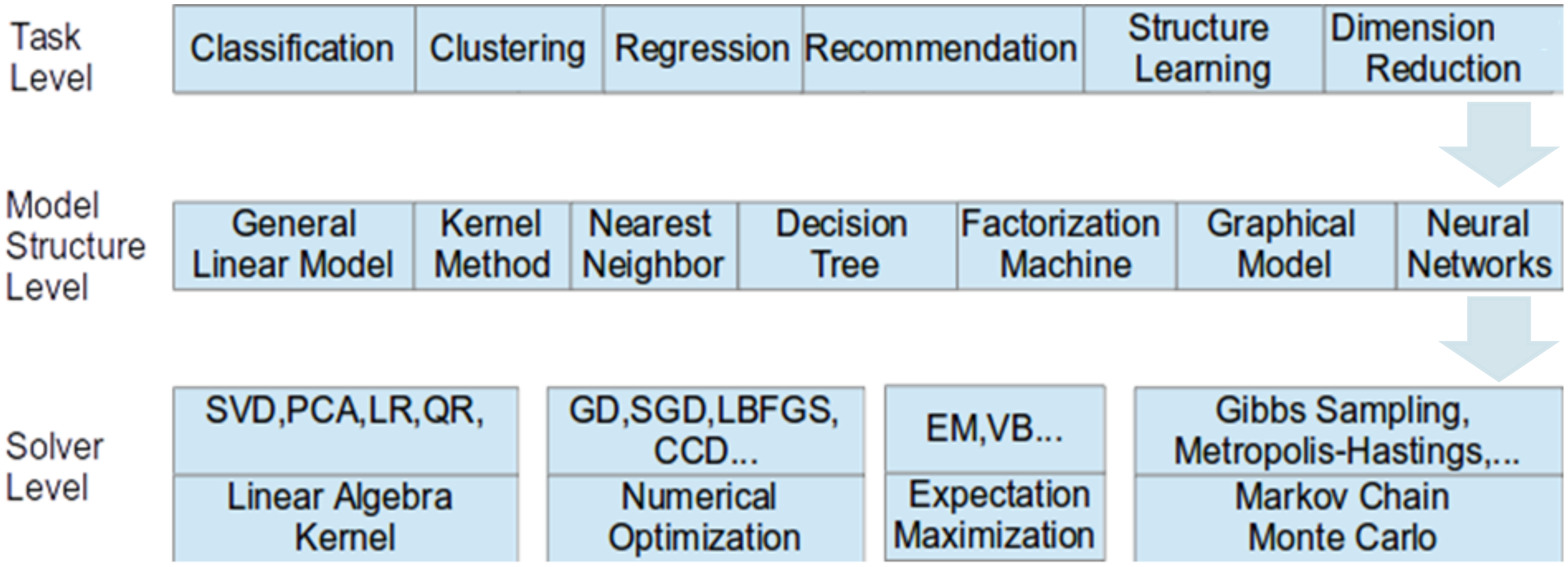
- getEvent
- waitEvent
- sendEvent

Collective Communication Operations

Operation Name	Algorithm	Time Complexity[a]
broadcast	chain	$n\beta$
	minimum spanning tree	$(\log_2 p)n\beta$
reduce	minimum spanning tree	$(\log_2 p)n\beta$
allgather	bucket	$pn\beta$
allreduce	bi-directional exchange	$(\log_2 p)n\beta$
regroup	point-to-point	$n\beta$
push & pull	point-to-point plus routing optimization	$n\beta$
rotate	exchange data between neighbors on a ring topology	$n\beta$

[a] Note in "time complexity", p is the number of processes, n is the number of input data items per worker, β is the per data item transmission time, communication startup time is neglected and the time complexity of the "point-to-point" based algorithms are estimated regardless of potential network conflicts.

Taxonomy for Machine Learning Algorithms



Optimization and related issues

- Task level only can't capture the traits of computation
- Model is the key for iterative algorithms. The structure (e.g. vectors, matrix, tree, matrices) and size are critical for performance
- Solver has specific computation and communication pattern

We investigate different computation and communication patterns of important ml algorithms

Parallel Machine Learning Application Implementation Guidelines

Application

- Latent Dirichlet Allocation, Matrix Factorization, Linear Regression…

Algorithm

- Expectation-Maximization, Gradient Optimization, Markov Chain Monte Carlo…

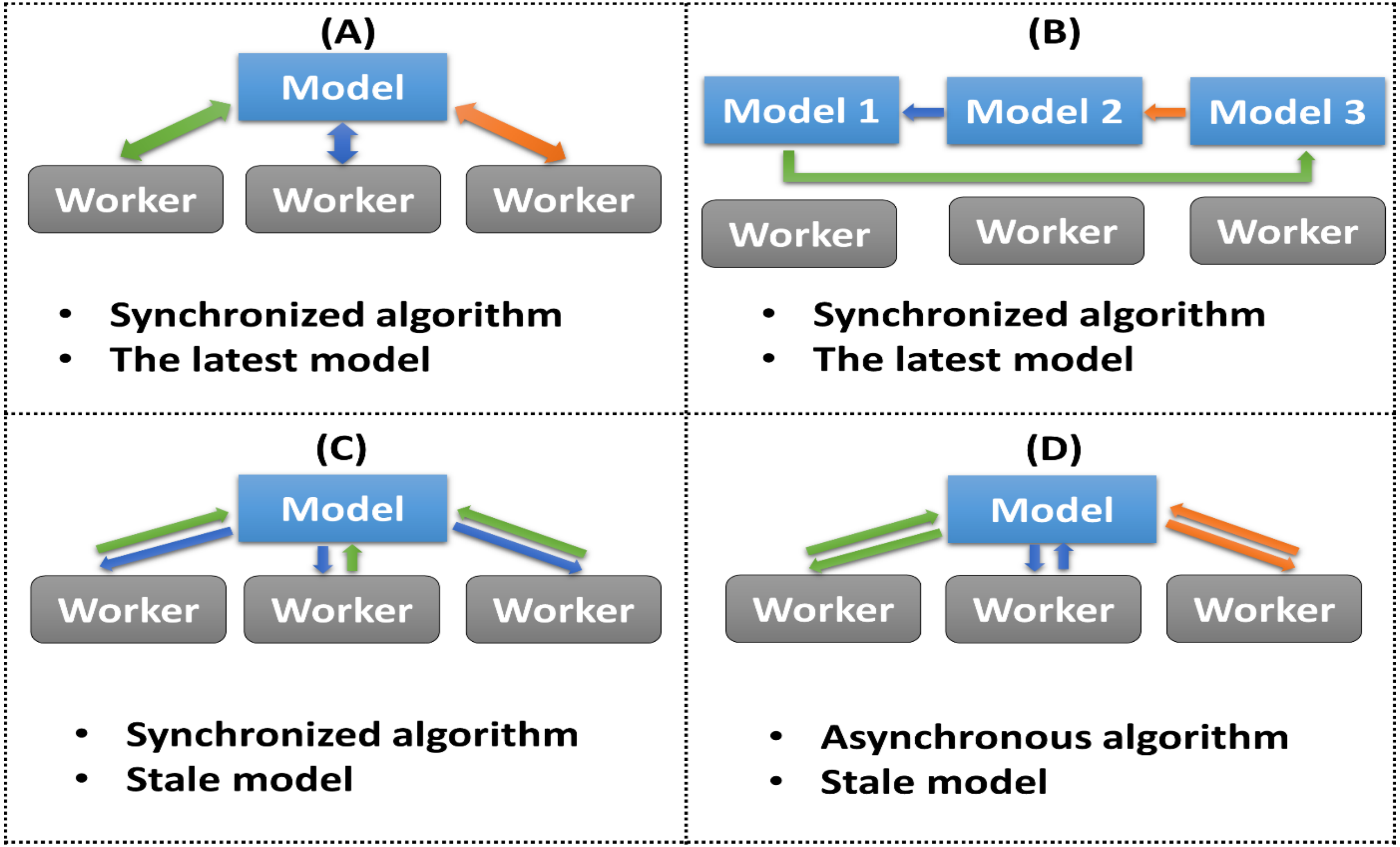
Computation Model

- Locking, Rotation, Allreduce, Asynchronous

System Optimization

- Collective Communication Operations
- Load Balancing on Computation and Communication
- Per-Thread Implementation

Computation Models

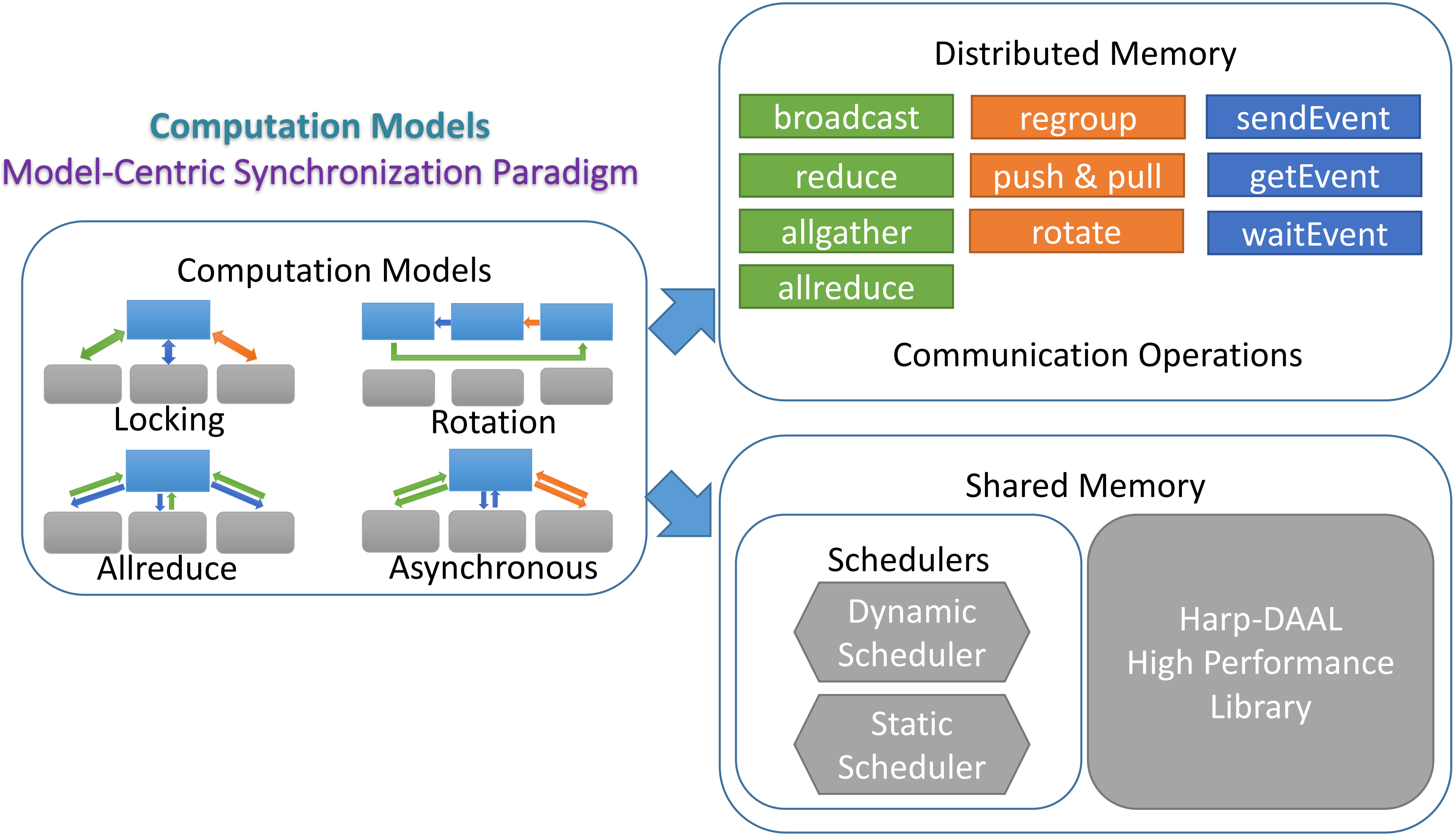


[8] B. Zhang, B. Peng, and J. Qiu, "Model-centric computation abstractions in machine learning applications," in Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, BeyondMR@SIGMOD 2016

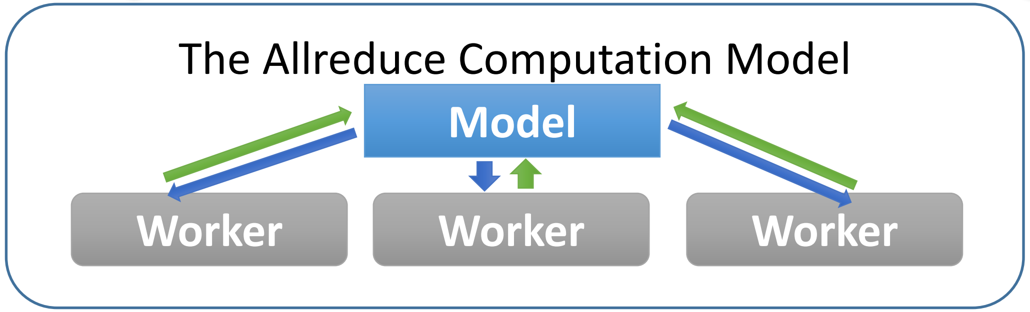
Harp Solution to Big Data Problems

Computation Models

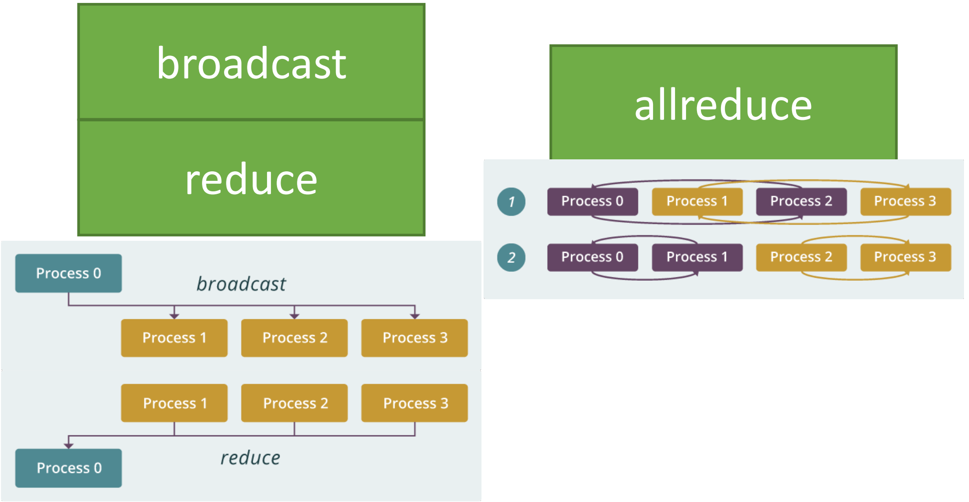
Model-Centric Synchronization Paradigm



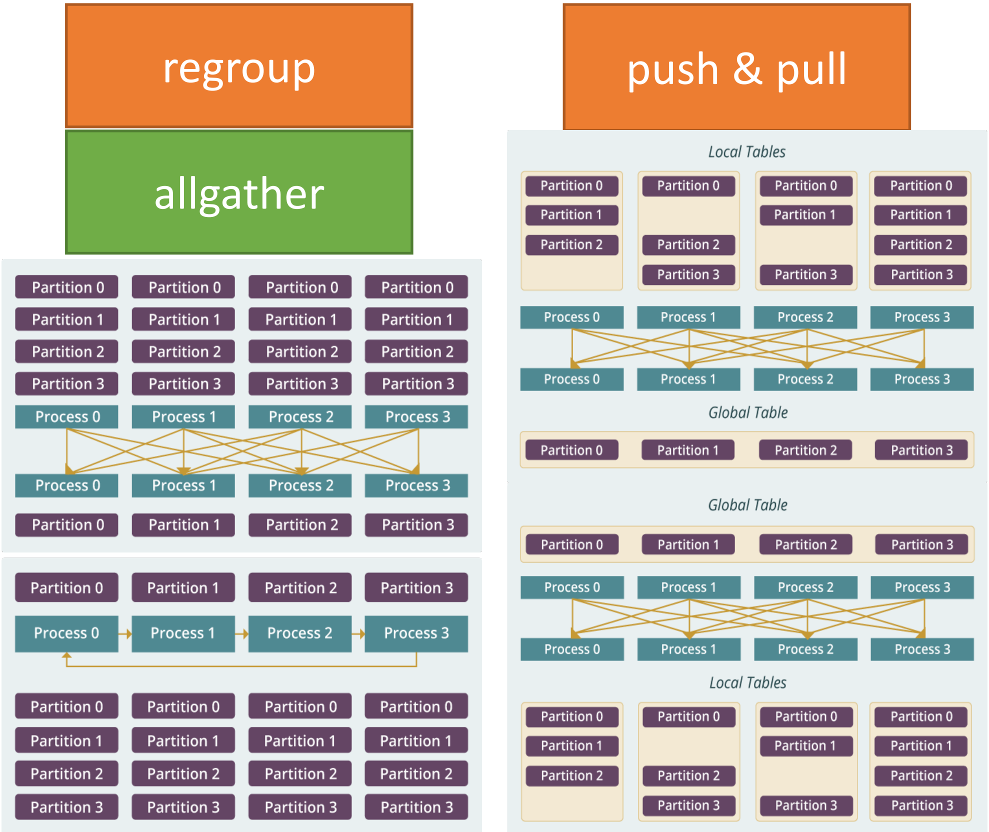
Example: K-means Clustering



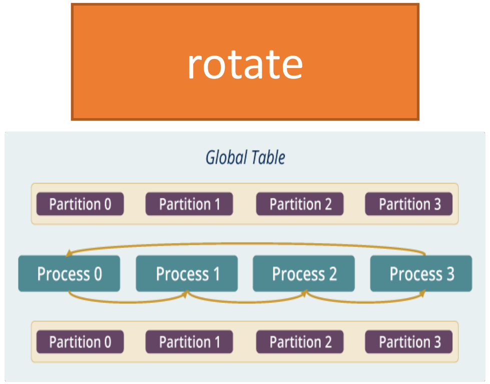
When the model size is small



When the model size is large but can still be held in each machine's memory



When the model size cannot be held in each machine's memory



Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work

Hardware specifications

Table I SPECIFICATION OF XEON PHI 7250 KNL

Cores		Memory		Node Spec		Misc Spec	
Cores	68	DDR4	190 GB	Network	Omni-path	Instruction Set	64 bit
Base Freq	1.4GHz	MCDRAM	16 GB	Peak Port Band	100 Gbps	IS Extension	AVX512
L1 Cache	2 MB	DDR4-Band	90 Gbps	Socket	1	Max Threads	271
L2 Cache	34 MB	MCDRAM-Band	400 Gbps	Disk	1 TB	VPUs	136

Table II Specification of Haswell Xeon E5 2699 v3

Cores		Memory		Node Spec		Misc Spec	
Cores	36	DDR4	130 GB	Network	InfiniBand	Instruction Set	64 bit
Base Freq	2.3GHz	HBM	none	Peak Port Band	56 Gbps	IS Extension	AVX2
L1/L2 Cache	32/256 KB	DDR4-Band	90 Gbps	Socket	2	Max Threads	72
L3 Cache	45 MB	HBM-Band	none	Disk	8 TB	VPUs	168

All scalability tests run on the above Haswell (128 node) and KNL (64 node) clusters.

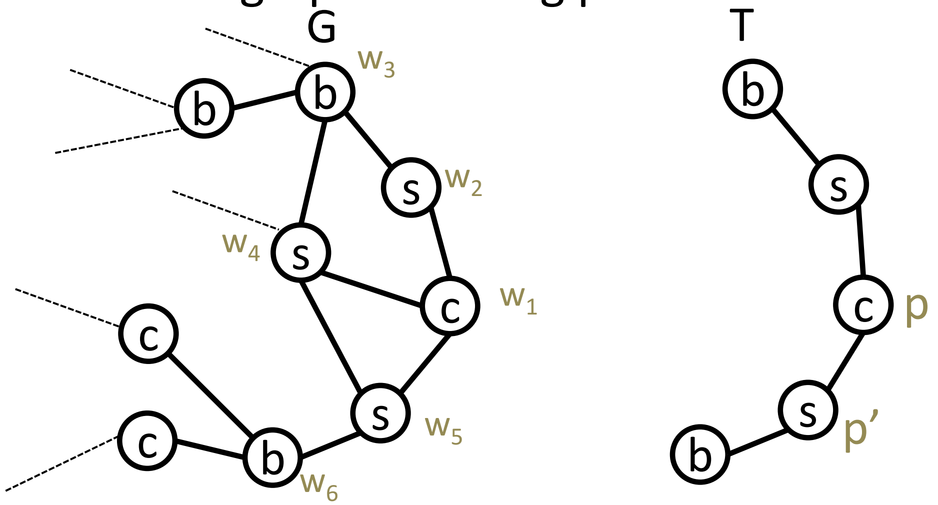
Harp-SAhad SubGraph Mining

VT and IU collaborative work

Relational sub-graph isomorphism problem: find sub-graphs in G which are isomorphic to the given template T.

SAhad is a challenging graph application that is both data intensive and communication intensive.

Harp-SAhad is an implementation for sub-graph counting problem based on SAHAD algorithm and Harp framework.



Network	No. Of Nodes (in million)	No. Of Edges (in million)	Size (MB)
Web-google	0.9	4.3	65
Miami	2.1	51.2	740
Nyc	18	480	7856

Table IV Networks of Graph Applications

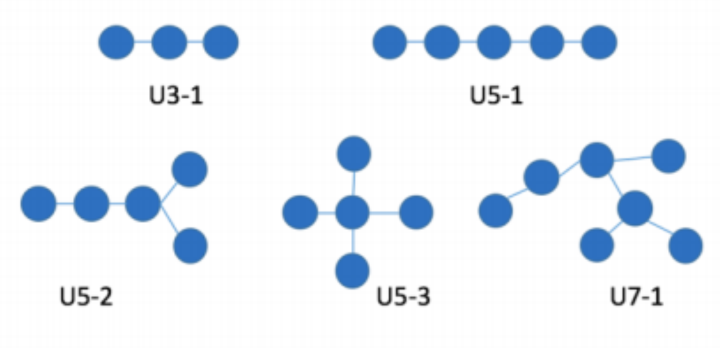


Figure Sub-graph Templates

[9] Zhao Z, Wang G, Butt A, Khan M, Kumar VS Anil, Marathe M. SAHad: Subgraph analysis in massive networks using hadoop. Shanghai, China: IEEE Computer Society; 2012:390–401. Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium.

Harp-SAHad Performance Results

VT and IU collaborative work

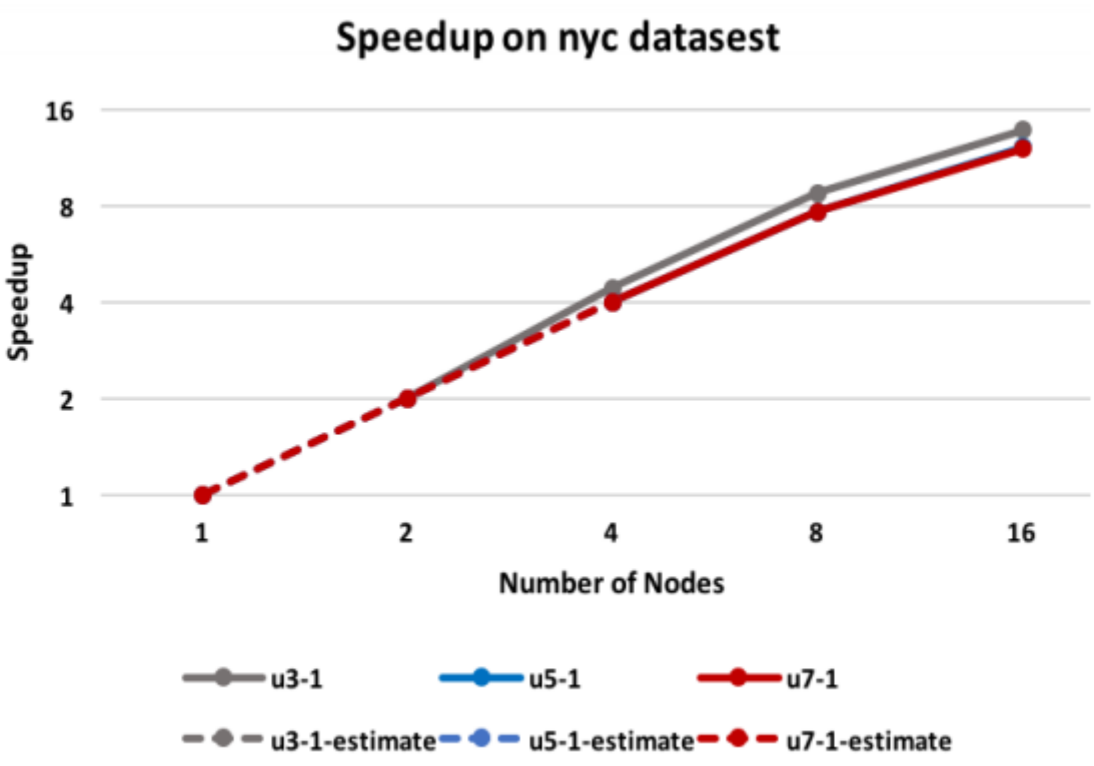


Figure Speedup on nyc dataset

Figure Running time break-down on u3-1 tempalte.

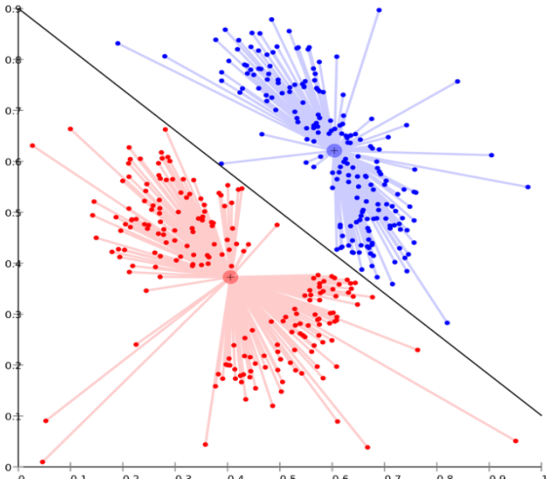
Test Plan and Datasets

Table III DATASETS USED IN K-MEANS, MF-SGD, AND ALS

Dataset	Kmeans-Single	Kmeans-Multi	Movielens	Netflix	Yahoomusic	Enwiki	Hugewiki
#Training	5000000	20000000	9301274	99072112	252800275	609700674	3074875354
#Test	none	none	698780	1408395	4003960	12437156	365998592
#centroid	10000	100000	none	none	none	none	none
Dim	100	100	40	40	100	100	1000
λ	none	none	0.05	0.05	1	0.01	0.01
γ	none	none	0.003	0.002	0.0001	0.001	0.004

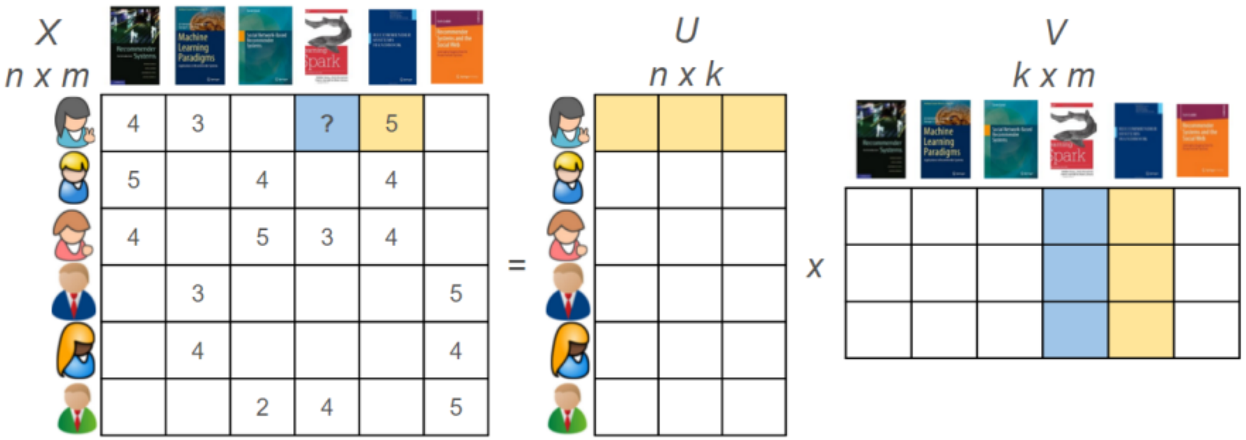
Harp-DAAL Applications

Harp-DAAL-Kmeans



Harp-DAAL-SGD

Harp-DAAL-ALS



Harp-DAAL-Kmeans:

- Clustering
- Vectorized computation
- **Small model data**
- Regular Memory Access

Harp-DAAL-SGD:

- Matrix Factorization
- Huge model data
- **Random Memory Access**
- Easy to scale up
- Hard to parallelize

Harp-DAAL-ALS:

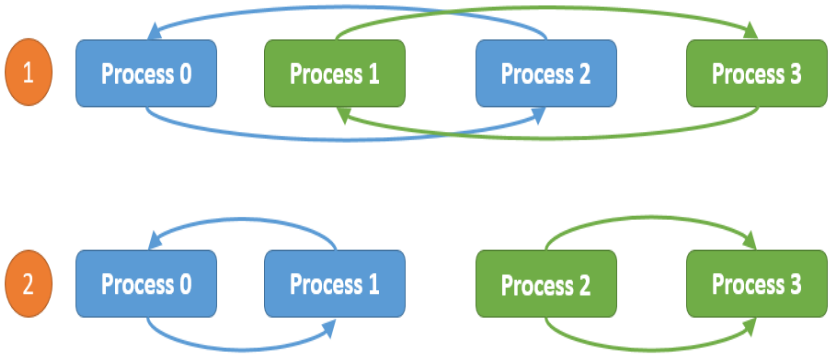
- Matrix Factorization
- Huge model data
- **Regular Memory Access**
- Easy to parallelize
- Hard to scale up

[10] Langshi Chen, Bo Peng, Bingjing Zhang, Tony Liu, Yiming Zou, Lei Jiang, Robert Henschel, Craig Stewart, Zhang Zhang, Emily Mccallum, Zahniser Tom, Omer Jon, Judy Qiu, Benchmarking Harp-DAAL: High Performance Hadoop on KNL Clusters, in the Proceedings of the International Conference on Cloud Computing (CLOUD 2017), June 25-30, 2017.

Computation models for K-means

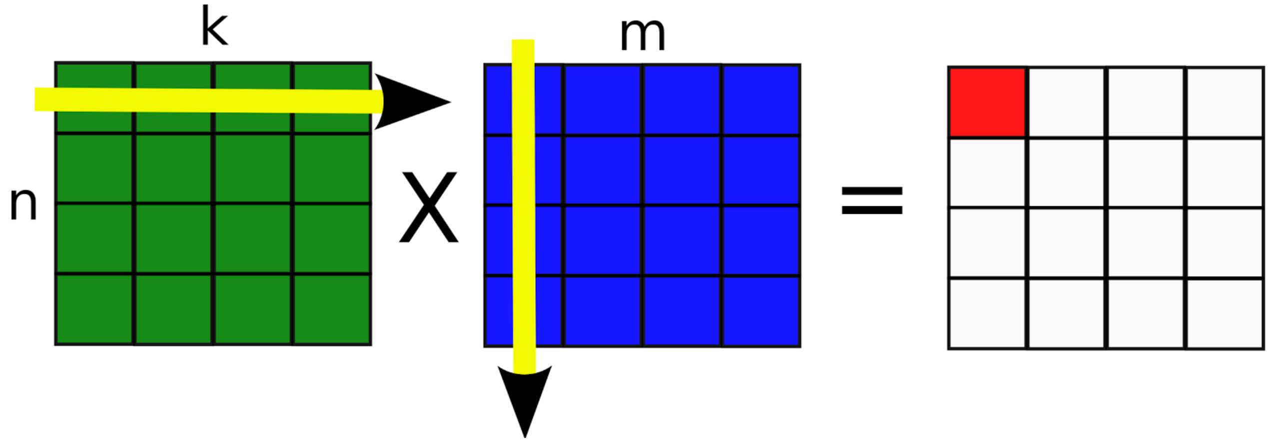
Harp-DAAL-Kmeans

- Inter-node: Allreduce, Easy to implement, efficient when model data is not large



- Intra-node: Shared Memory, matrix-matrix operations, xGemm: aggregate vector-vector distance computation into matrix-matrix multiplication, higher computation intensity (BLAS-3)

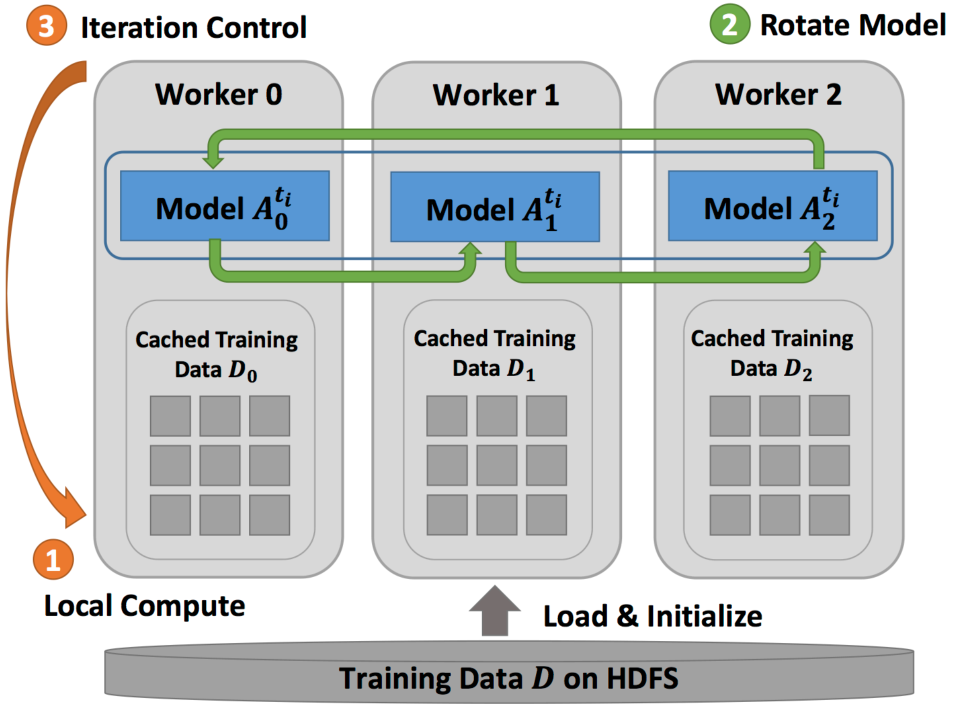
procedure
Given $(x^1, x^2, \ldots, x^m)$, $\forall i, x^i \in \mathbb{R}^n$
Initialize centroids randomly: $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$
Repeat until convergence
$\forall i,\ c^i := argmin_j \|x^i - \mu_j\|^2$
$\forall j,\ \mu_j := \frac{\sum_{i=1}^{m} 1\{c^i = j\} x^i}{\sum_{i=1}^{m} 1\{c^i = j\}}$
End Repeat
end procedure



$$C \leftarrow \alpha op(A) op(B) + \beta C$$

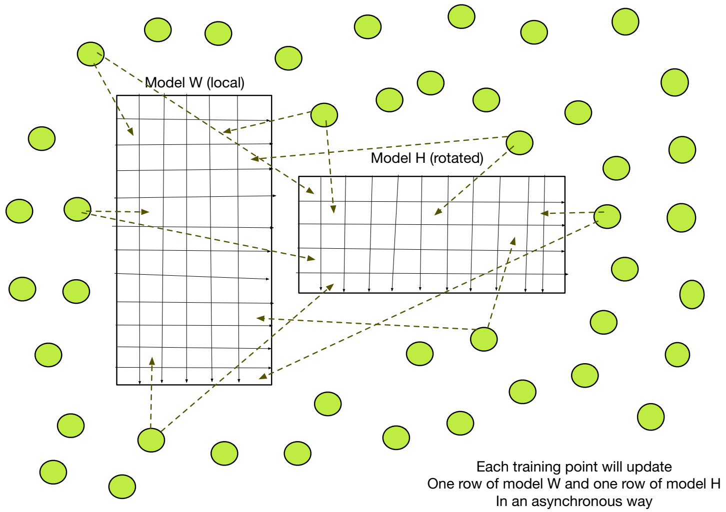
Computation models for MF-SGD

- Inter-node: Rotation
- Intra-node: Asynchronous



Rotation: Efficent when the mode data
Is large, good scalability

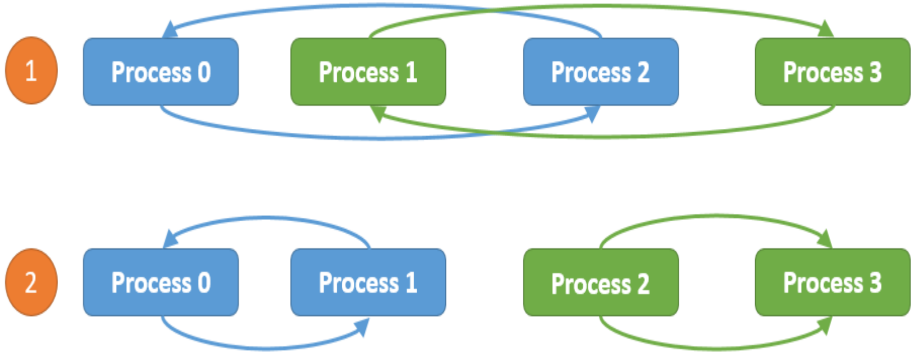
procedure
  $R \in \mathbb{R}^{m\times n}$, $P \in \mathbb{R}^{k\times m}$, and $Q \in \mathbb{R}^{k\times n}$
  while true **do**
    select randomly a point r_{ij} from R
    $e_{ij} = r_{ij} - p_i^T q_j$
    $p_i \leftarrow p_i + \gamma(e_{ij} q_j - \lambda_P p_i$
    $q_j \leftarrow q_j + \gamma(e_{ij} p_i - \lambda_Q q_j$
    if P, Q converged **then**
      Exit While loop
    end if
  end while
end procedure



Asynchronous: Random access to model data
Good for thread-level workload balance.

Computation Models for ALS

- Inter-node: Allreduce

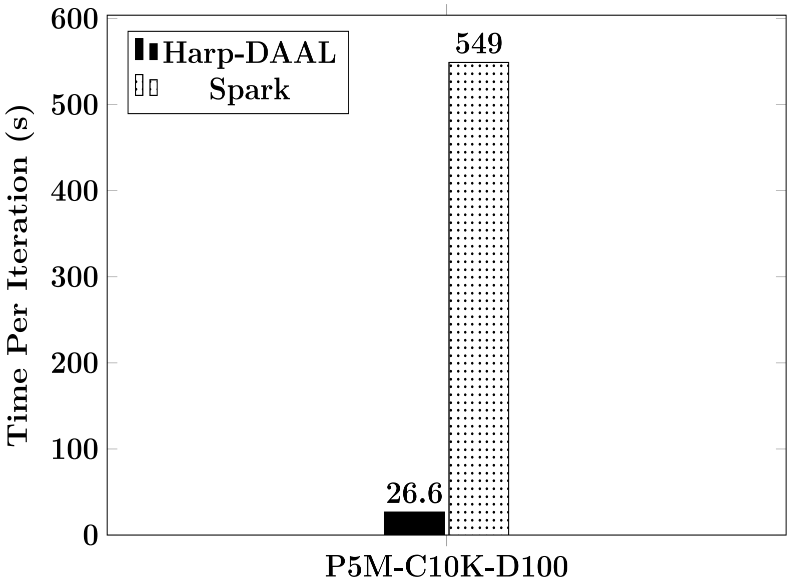


Intra-node: Shared Memory, Matrix operations
xSyrk: symmetric rank-k update

$$C \leftarrow \alpha A A^T + \beta C$$
$$A \leftarrow \alpha x x^T + A$$

procedure
  Load R, R^T
  Initialize X, Y
  repeat
    for $i = 1, 2, \ldots, n$ **do**
      $V_i = Y_{I_i} R^T(i, I_i)$
      $A_i = Y_{I_i} Y_{I_i}^T + \lambda n_{x_i} E$
      $x_i = A_i^{-1} V_i$
    end for
    for $j = 1, 2, \ldots, m$ **do**
      $U_j = X_{I_j} R(I_j, j)$
      $B_j = X_{I_j} X_{I_j}^T + \lambda n_{m_j} E$
      $y_j = B_j^{-1} U_j$
    end for
  until convergence
end procedure

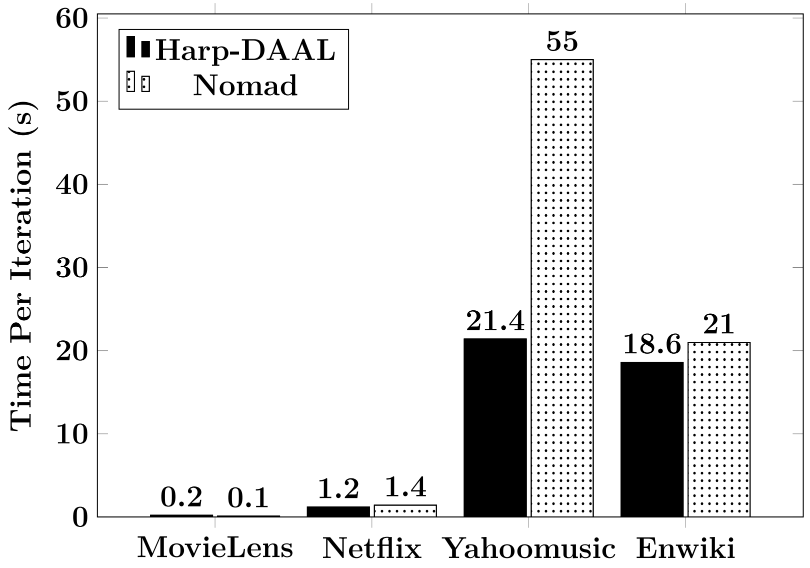
Performance on KNL Single Node



Harp-DAAL-Kmeans vs. Spark-Kmeans:

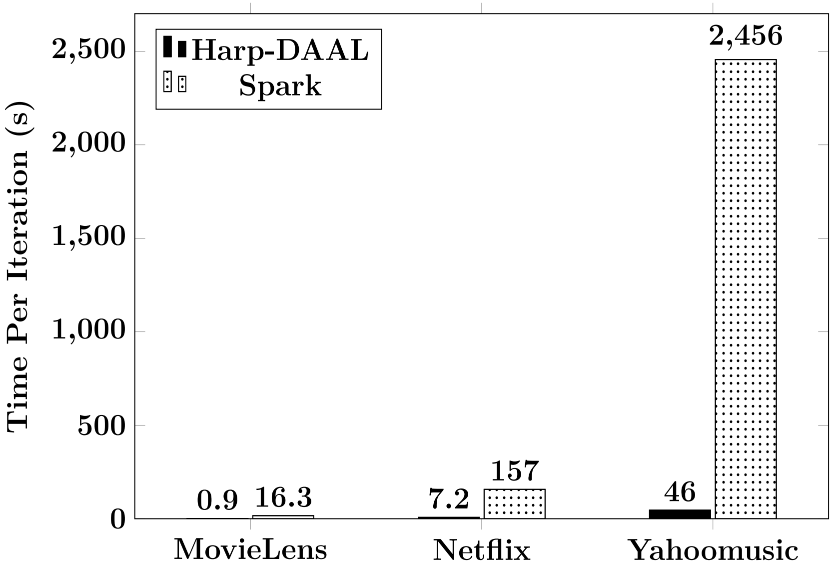
~ 20x speedup

1) Harp-DAAL-Kmeans invokes MKL matrix operation kernels at low level
2) Matrix data stored in contiguous memory space, leading to regular access pattern and data locality



Harp-DAAL-SGD vs. NOMAD-SGD

1) Small dataset (MovieLens, Netflix): comparable perf
2) Large dataset (Yahoomusic, Enwiki): ***1.1x to 2.5x***, depending on data distribution of matrices



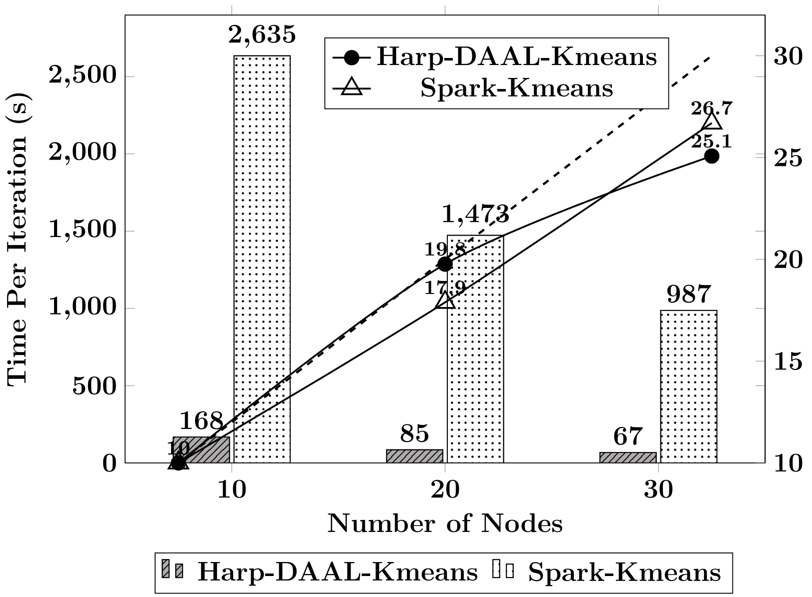
Harp-DAAL-ALS vs. Spark-ALS

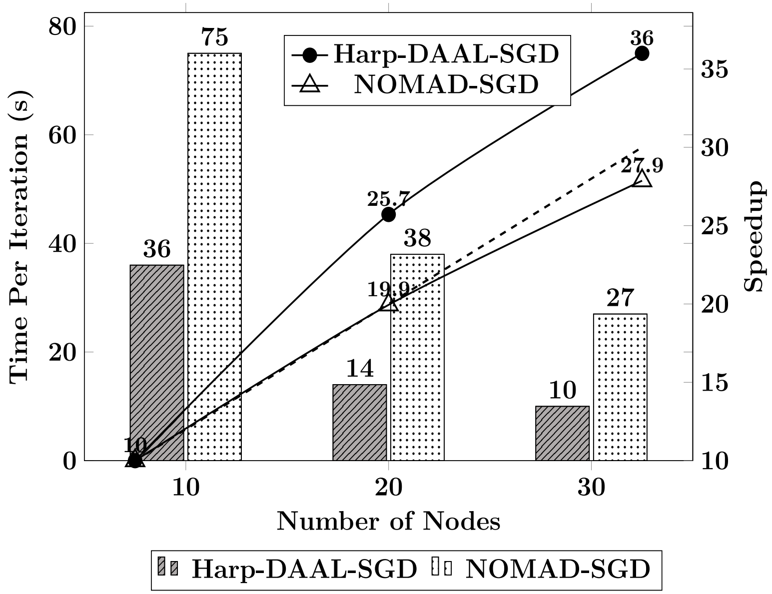
20x to 50x speedup

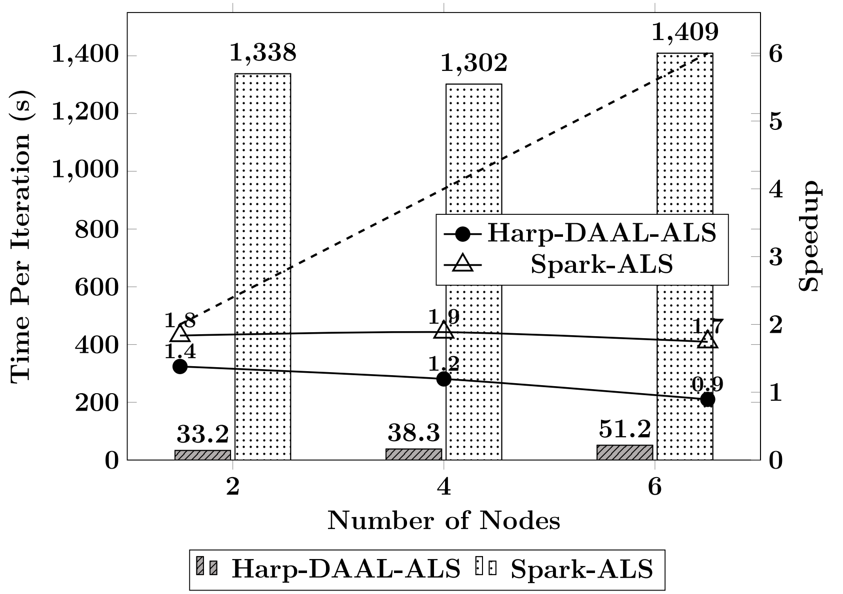
1) Harp-DAAL-ALS invokes MKL at low level
2) Regular memory access, data locality in matrix operations

Harp-DAAL has much better single node performance than Java solution (Spark-Kmeans, Spark-ALS) and comparable performance to state-of-arts C++ solution (NOMAD-SGD)

Performance on KNL Multi-Nodes







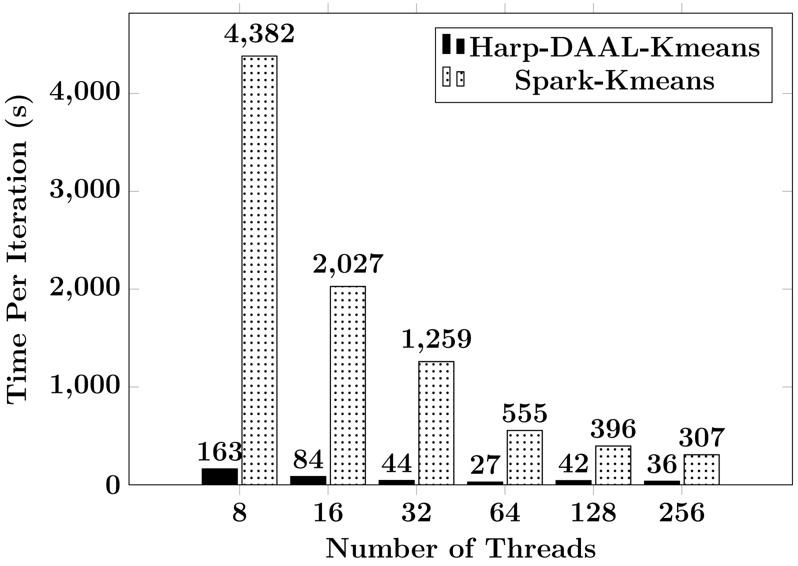
Harp-DAAL-Kmeans:
15x to 20x speedup over Spark-Kmeans
1) Fast single node performance
2) Near-linear strong scalability from 10 to 20 nodes
3) After 20 nodes, insufficient computation workload leads to some loss of scalability

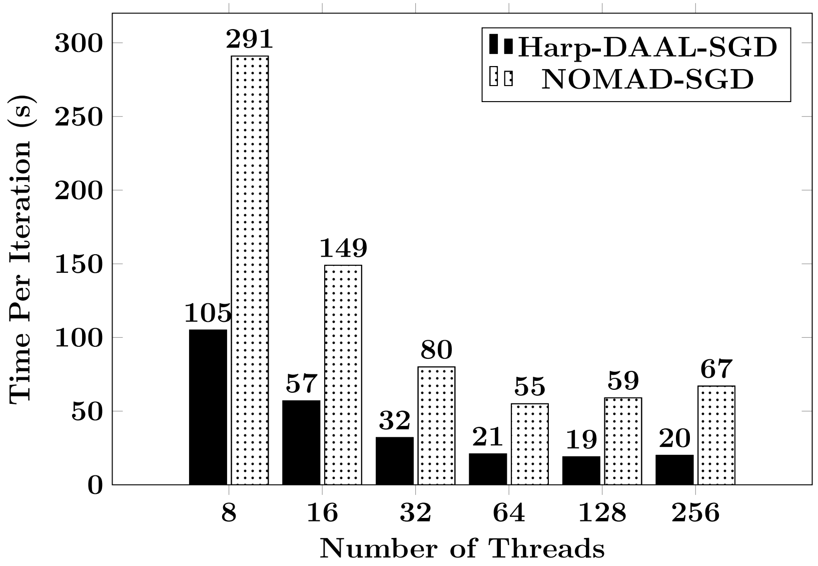
Harp-DAAL-SGD:
2x to 2.5x speedup over NOMAD-SGD
1) Comparable or fast single node performance
2) Collective communication operations in Harp-DAAL outperform point-to-point MPI communication in NOMAD

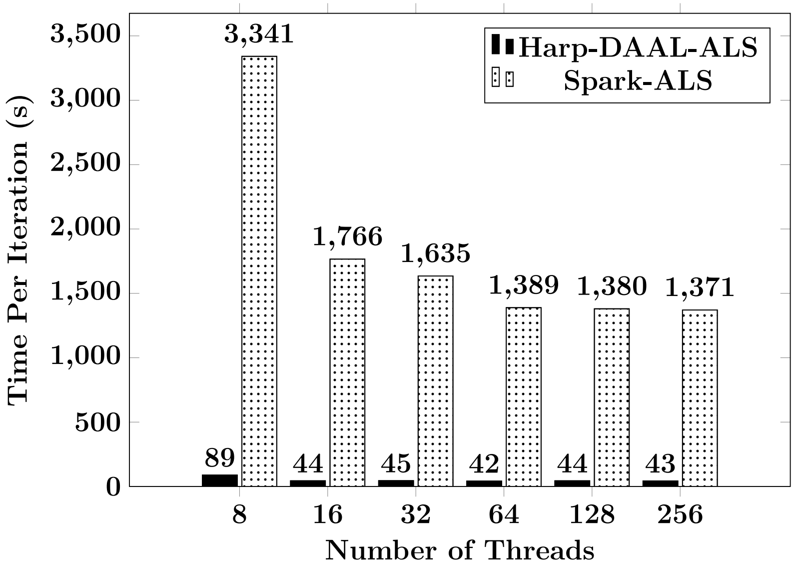
Harp-DAAL-ALS:
25x to 40x speedup over Spark-ALS
1) Fast single node performance
2) ALS algorithm is not scalable (high communication ratio)

Harp-DAAL combines the benefits from local computation (DAAL kernels) and communication operations (Harp), which is much better than Spark solution and comparable to MPI solution.

Breakdown of Intra-node Performance



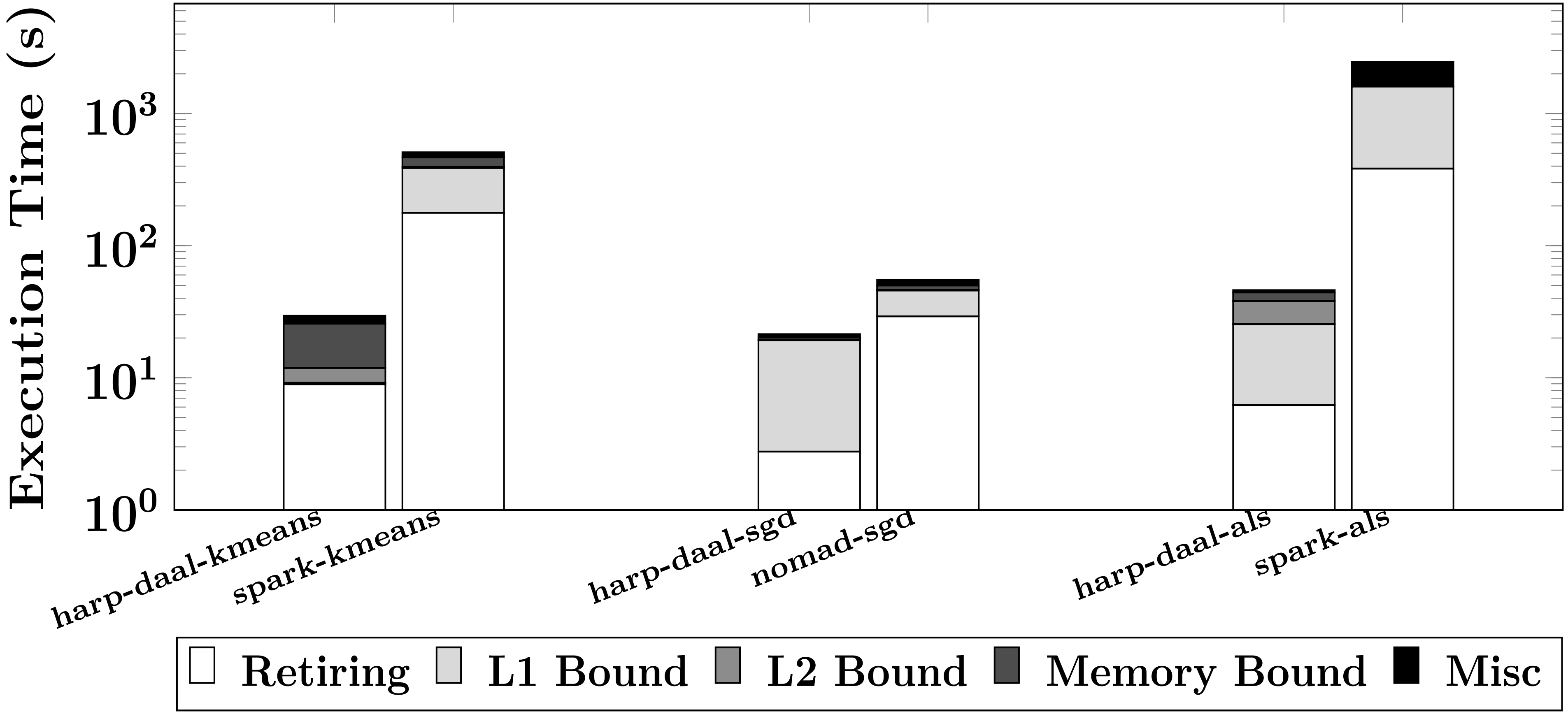




Thread scalability:

- Harp-DAAL best threads number: 64 (K-means, ALS) and 128 (MF-SGD), more than 128 threads no performance gain
  - communications between cores intensify
  - cache capacity per thread also drops significantly
- Spark best threads number 256, because Spark could not fully Utilize AVX-512 VPUs
- NOMAD-SGD could use AVX VPU, thus has 64 its best thread as that of Harp-DAAL-SGD

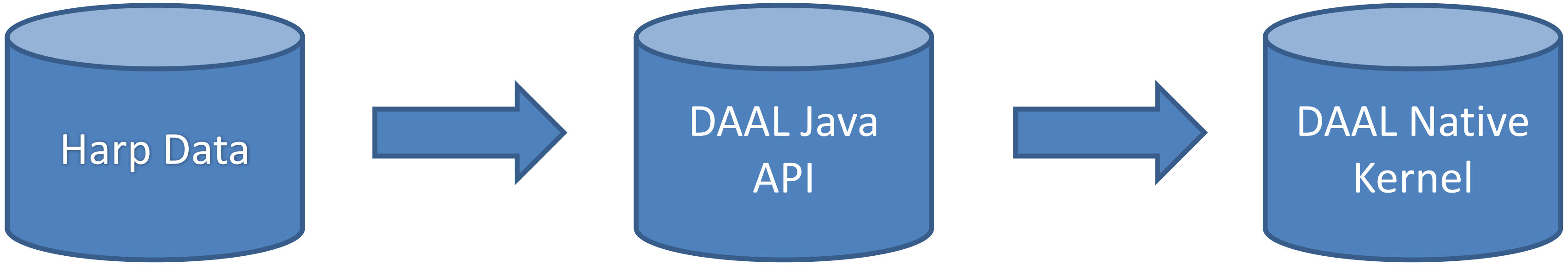
Breakdown of Intra-node Performance



Spark-Kmeans and Spark-ALS dominated by Computation (retiring), no AVX-512 to reduce retiring Instructions, Harp-DAAL improves L1 cache bandwidth utilization due to AVX-512

Code Optimization Highlights

Data Conversion



Harp Data

- Table<Obj>
- Data on JVM Heap

DAAL Java API

- NumericTable
- Data on JVM heap
- Data on Native Memory

DAAL Native Kernel

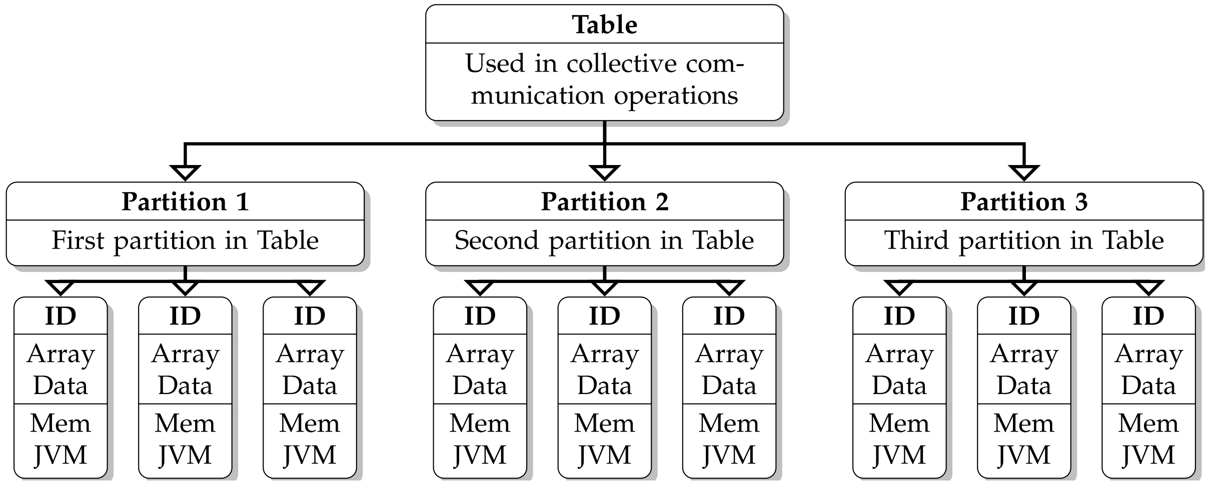
- MicroTable
- Data on Native Memory

A single DirectByteBuffer has a size limite of 2 GB

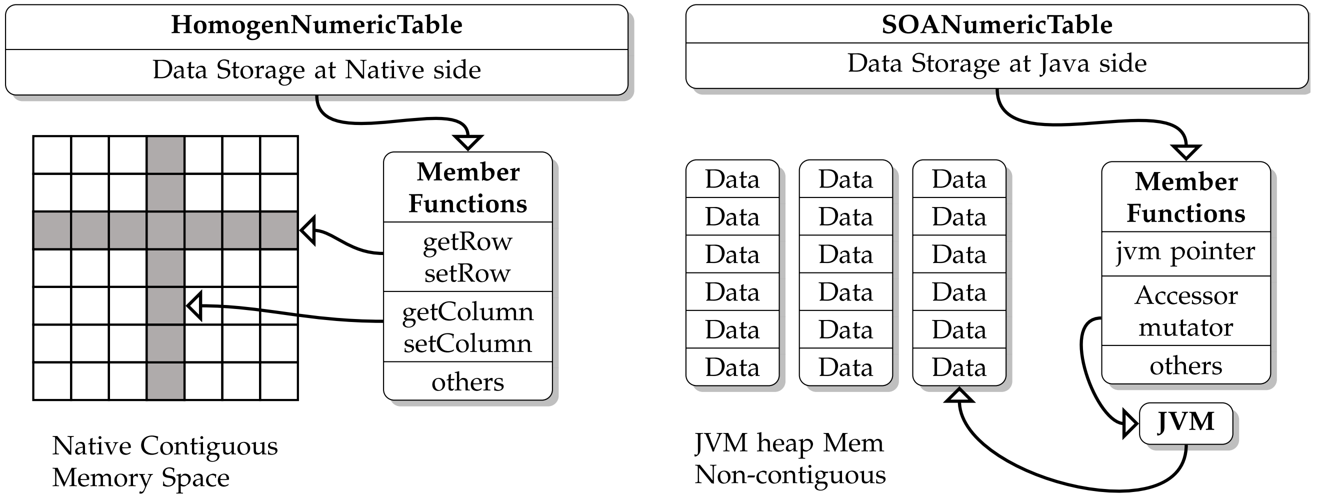
Two ways to store data using DAAL Java API

- Keep Data on JVM heap
    - no contiguous memory access requirement
    - Small size DirectByteBuffer and parallel copy (OpenMP)
- Keep Data on Native Memory
    - contiguous memory access requirement
    - Large size DirectByteBuffer and bulk copy

Data Structures of Harp & Intel's DAAL



Harp Table consists of Partitions



DAAL Table has different types of Data storage

Table<Obj> in Harp has a three-level data Hierarchy

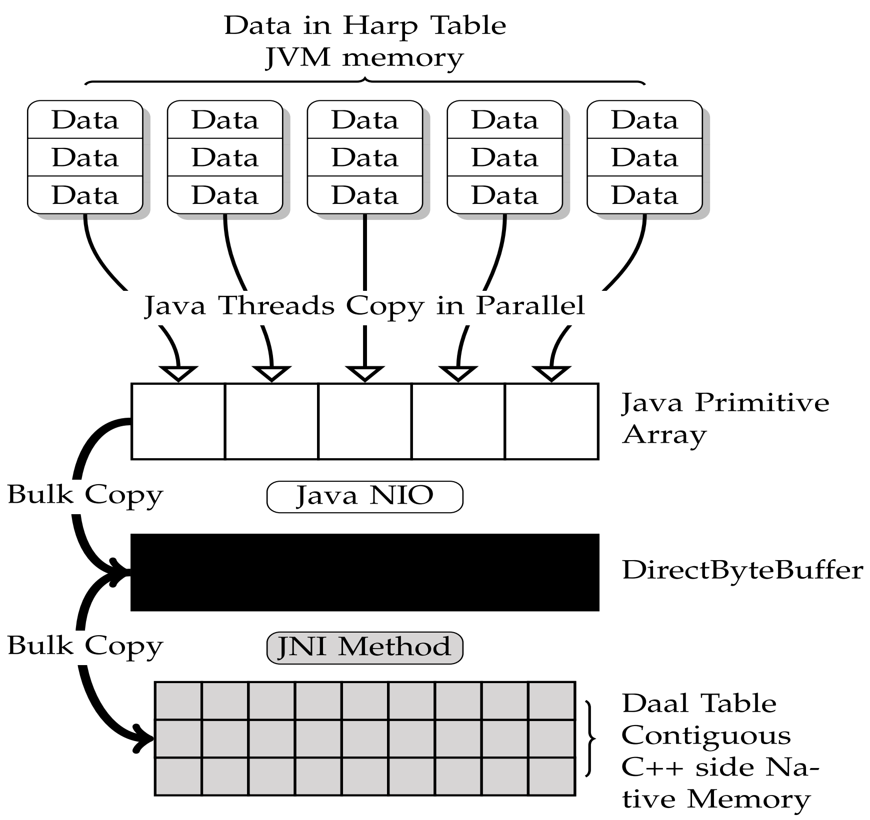
- Table: consists of partitions
- Partition: partition id, container
- Data container: wrap up Java objs, primitive arrays

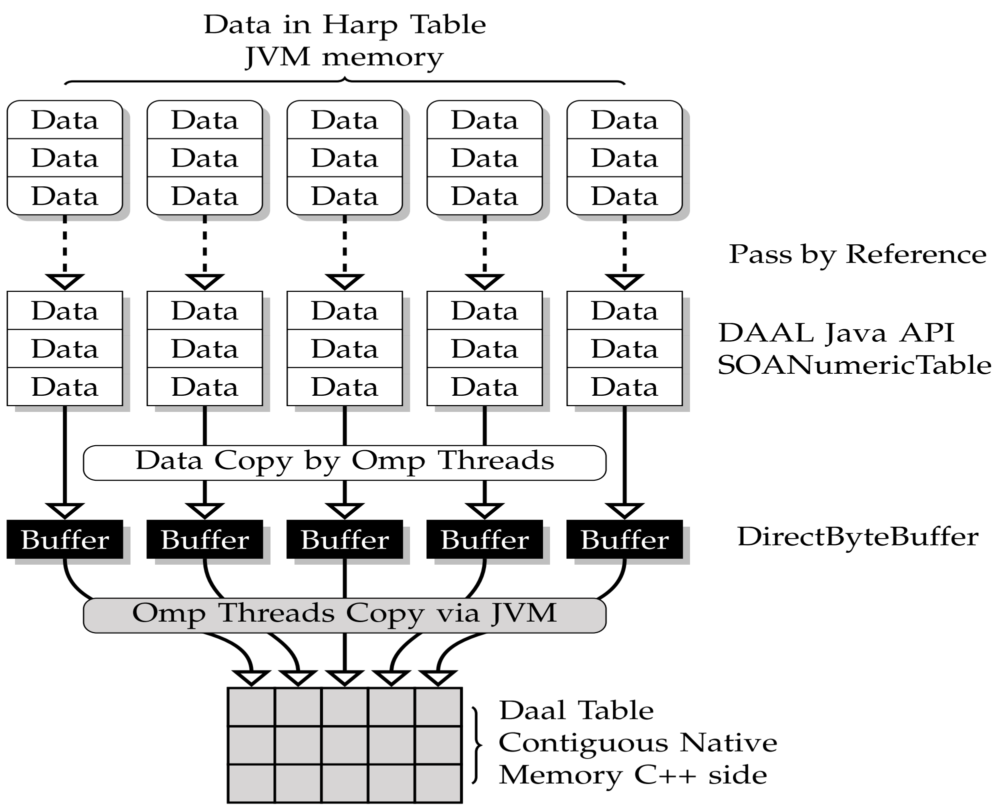
Data in different partitions, non-contiguous in memory

NumericTable in DAAL stores data either in Contiguous memory space (native side) or non-contiguous arrays (Java heap side)

Data in contiguous memory space favors matrix operations with regular memory accesses.

Two Types of Data Conversion





JavaBulkCopy:

Dataflow: Harp Table<Obj> -----
Java primitive array ---- DiretByteBuffer ----
NumericTable (DAAL)

Pros: Simplicity in implementation

Cons: high demand of DirectByteBuffer size

NativeDiscreteCopy:

Dataflow: Harp Table<Obj> ----
DAAL Java API (SOANumericTable)
---- DirectByteBuffer ---- DAAL native memory

Pros: Efficiency in parallel data copy

Cons: Hard to implement at low-level kernels

Outline

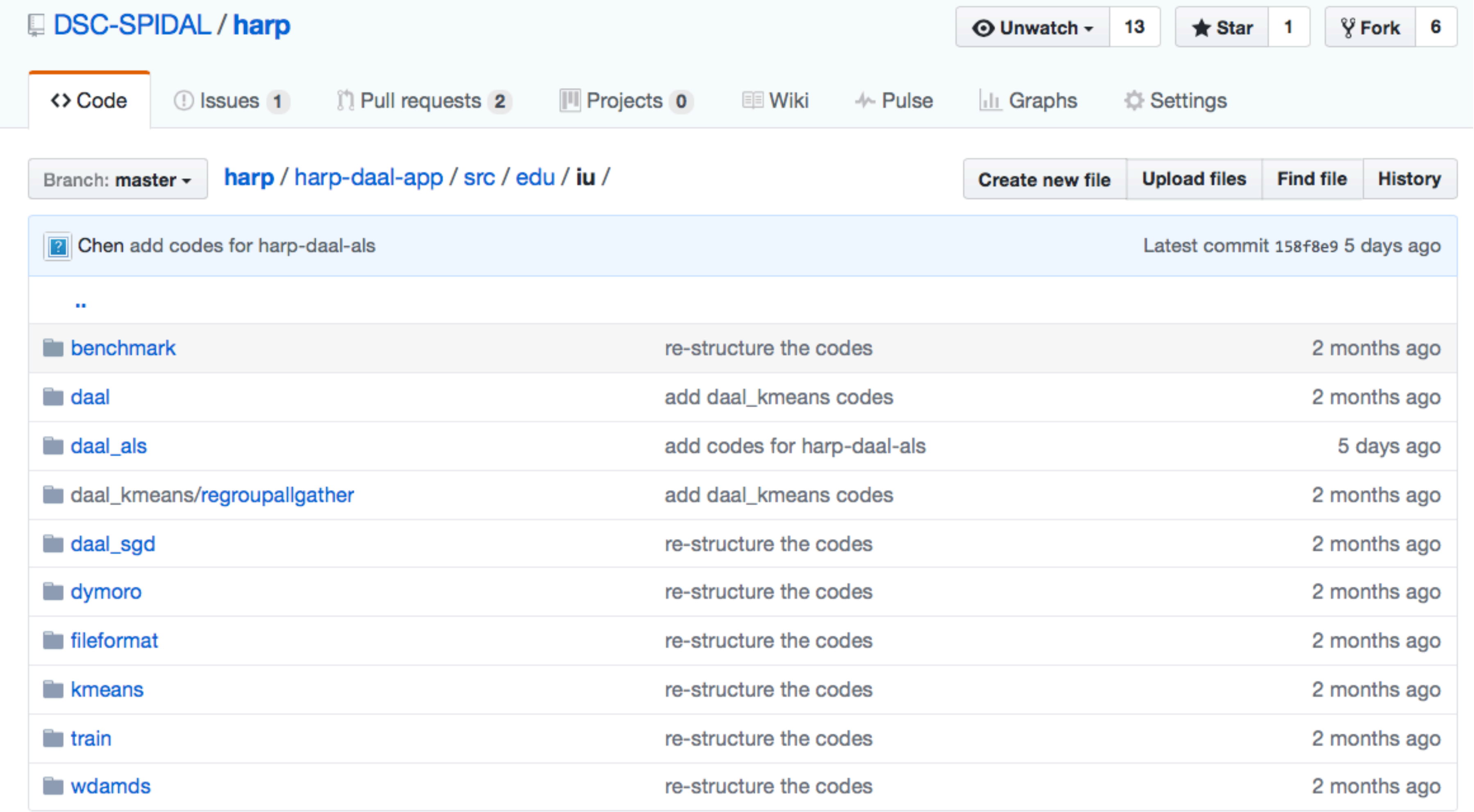
1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work

Conclusions

- 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
  - Locking, Rotation, Allreduce, Asynchroneous
- **HPC-ABDS:** High performance **Hadoop** (with Harp-DAAL) on KNL and Haswell clusters

Hadoop/Harp-DAAL: Prototype and Production Code

Open Source Available at https://dsc-spidal.github.io/harp



Source codes became available on Github in February, 2017.

- Harp-DAAL follows the same standard of DAAL's original codes
- Six Applications
  - Harp-DAAL Kmeans
  - Harp-DAAL MF-SGD
  - Harp-DAAL MF-ALS
  - Harp-DAAL SVD
  - Harp-DAAL PCA
  - Harp-DAAL Neural Networks

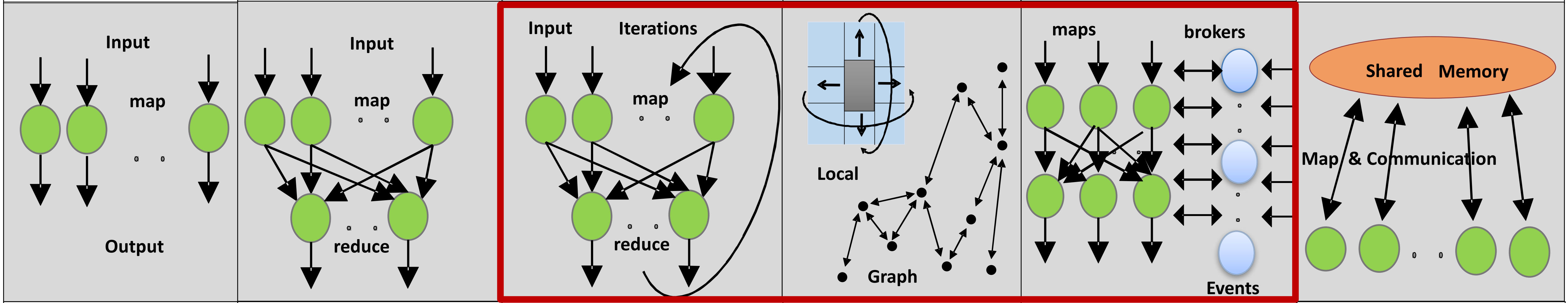
Scalable Algorithms implemented using Harp

Algorithm	Category	Applications	Features	Computation Model	Collective Communication
K-means	Clustering	Most scientific domain	Vectors	AllReduce	allreduce, regroup+allgather, broadcast+reduce, push+pull
				Rotation	rotate
Multi-class Logistic Regression	Classification	Most scientific domain	Vectors, words	Rotation	regroup, rotate, allgather
Random Forests	Classification	Most scientific domain	Vectors	AllReduce	allreduce
Support Vector Machine	Classification, Regression	Most scientific domain	Vectors	AllReduce	allgather
Neural Networks	Classification	Image processing, voice recognition	Vectors	AllReduce	allreduce
Latent Dirichlet Allocation	Structure learning (Latent topic model)	Text mining, Bioinformatics, Image Processing	Sparse vectors; Bag of words	Rotation	rotate, allreduce
Matrix Factorization	Structure learning (Matrix completion)	Recommender system	Irregular sparse Matrix; Dense model vectors	Rotation	rotate
Multi-Dimensional Scaling	Dimension reduction	Visualization and nonlinear identification of principal components	Vectors	AllReduce	allgarther, allreduce
Subgraph Mining	Graph	Social network analysis, data mining, fraud detection, chemical informatics, bioinformatics	Graph, subgraph	Rotation	rotate
Force-Directed Graph Drawing	Graph	Social media community detection and visualization	Graph	AllReduce	allgarther, allreduce

Future Work

- **Harp-DAAL** machine learning and data analysis applications with optimal performance.
- **Online Clustering** with **Harp** or **Storm** integrates parallel and dataflow computing models
- Start HPC Cloud incubator project in Apache to bring HPC-ABDS to community

Six Computation Paradigms for Data Analytics



(1) Map Only Pleasingly Parallel	(2) Classic Map-Reduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication	(5) Map-Streaming	(6) Shared memory Map-Communication
- BLAST Analysis - Local Machine Learning - Pleasingly Parallel	- High Energy Physics (HEP) Histograms, - Web search - Recommender Engines	- Expectation Maximization - Clustering - Linear Algebra - PageRank	- Classic MPI - PDE Solvers and Particle Dynamics - Graph	- Streaming images from Synchrotron sources, Telescopes, Internet of Things	- Difficult to parallelize - asynchronous parallel Graph

These 3 Paradigms are our Focus

Acknowledgements

We gratefully acknowledge support from NSF, IU and Intel Parallel Computing Center (IPCC) Grant.

Langshi Cheng Bingjing Zhang Bo Peng Kannan Govindarajan

Supun Kamburugamuve Yiming Zhou Ethan Li Mihai Avram Vibhatha Abeykoon

Intelligent Systems Engineering
School of Informatics and Computing
Indiana University