Introduction to Harp: when Big Data Meets HPC

Bingjing Zhang, Bo Peng, Langshi Chen, Ethan Li, Yiming Zhou, Judy Qiu
School of Informatics and Computing
Indiana University

Harp

Data analytics is undergoing a revolution in many scientific domains, demanding cost-effective parallel data analysis techniques. We consider the challenges of creating a high performance data analysis software framework in the context of the current HPC-ABDS software stack (High Performance Computing enhanced Apache Big Data Stack) [1]. We have summarized a list of current data processing software from either HPC or commercial sources [2]. Many critical components of the commodity stack (such as Hadoop) come from Apache open source projects for community usage, while HPC (such as collective communication) is needed to bring performance and other parallel computing capabilities.

Many machine learning algorithms are built on iterative computation, which can be formulated as

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

where *D* is the observed dataset, *A* is model parameters to learn, and *F* is the model update function. The algorithm keeps updating model *A* until convergence, either by reaching a threshold criterion or fixed number of iterations. There are several advantages of this iterative procedure as apparently simple functions can iterate and produce complex behavior for interesting problems. The power of iteration and its extensions lies in the approximation or accuracy that can be obtained at each step even if the computation stops abruptly before converges to the final answer.

To effectively support large-scale data processing, Twister [3] introduced iterative MapReduce using long-running processes or threads with in-memory caching of invariant data. Harp [4] introduces full collective communication in Table 1 (broadcast, reduce, allgather, allreduce, rotation, regroup or push & pull), adding a separate communication abstraction where the Harp prototype implements the MapCollective concept as a plug-in to Hadoop Ecosystem (see Figure 1 and Figure 2). Instead of using the shuffling phase, Harp uses optimized collective communication operations for data movement since fine-grained data alignment for multiple models is critical for improving performance. It further provides high-level interfaces with various synchronization patterns for parallelizing iterative computation. These enhancements make it possible to exploit HPC capabilities for big data software systems.

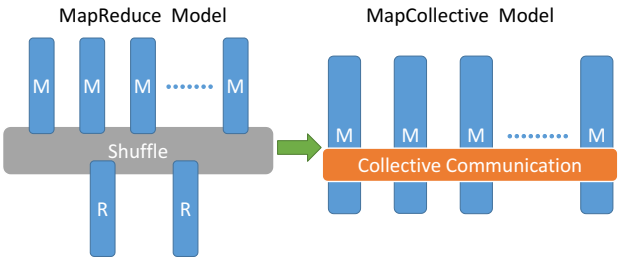


Figure 1 Map-Collective Model

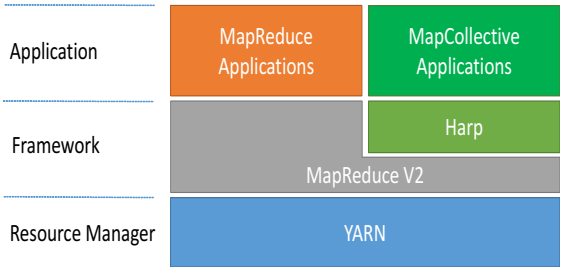


Figure 2 Harp Architecture

In order to build a general machine learning framework, we have studied different parallel patterns (kernels) of machine learning applications, looking in particular at Gibbs Sampling [Gibbs Sampling], Stochastic Gradient Descent (SGD) [5], Cyclic Coordinate Descent (CCD) [6] and K-Means clustering. These algorithms are fundamental for large-scale data analysis and cover several important categories: Markov Chain Monte Carlo (MCMC), Gradient Descent and Expectation and Maximization (EM). We show that parallel iterative algorithms can be categorized into four types of computation models (a) locking, (b) rotation, (c) allreduce, (d) asynchronous, based on the synchronization patterns and the effectiveness of the model parameter update.

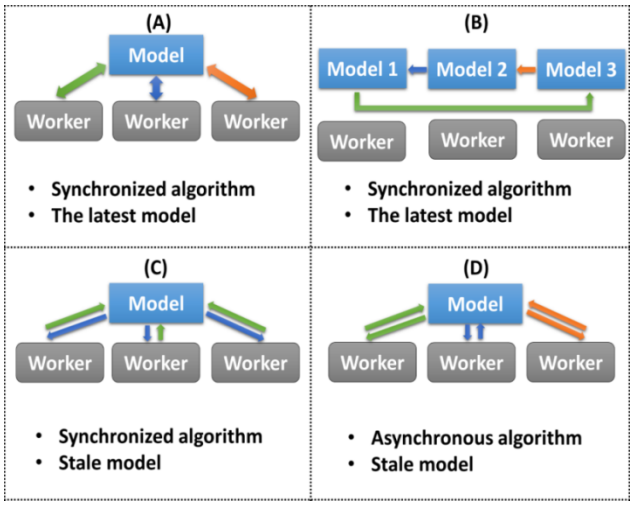


Figure 3 Computation Models

Based on these computation models and the collective communication abstractions defined, we propose a new set of model-centric computation abstractions shown in Figure 3. These set up parallel machine learning as the combination of training data-centric and model parameter-centric processing. Model A is "locking" based computation model while Model D is an "asynchronous" model that allows workers (mappers) to fetch or update the same model parameters in parallel with asynchronous "push" and "pull." This contrasts with Model B "allreduce" and Model C "rotation," which have a clear synchronization barrier. The Allreduce model is effective for machine learning algorithms with a summation form, but it doesn't scale for a problem with a model that cannot fit into the memory. Since rotation partitions the global model among distributed workers, it effectively reduces the memory footprint but requires model synchronization using collective communication.

The computation models provide the basis for a systematic approach to parallelizing iterative algorithms, which can be implemented on different programming interfaces (e.g. Harp, Petuum, MPI, Spark, Flink) and implementation details (e.g. Java, C++ languages). These tools differ in the computation models they support effectively. For example, "rotation" has been supported by Harp and Petuum, while "broadcast," "reduce" or "allreduce" have been implemented by most big data processing tools. Note that although the 5 Harp collectives ("broadcast, reduce, allgather, allreduce, rotation/shift") correspond to MPI operations since the communications are between processes (threads for Harp), "regroup" and "push&pull" are specific to data-driven solutions where a key ID of the data object decides the routing pattern.

Table 1: Collective Communication Operation Interfaces

Operation	Definition	Algorithm	Time
broadcast	The master worker broadcasts all the partitions to the tables on other workers.	chain	$n\beta$
		minimum spanning tree	

reduce	The partitions from all the workers are reduced to the master worker (partitions with the same ID are combined).	minimum spanning tree	*(log2 p)nβ*
allreduce	The partitions from all the workers are reduced and received by all the workers.	bi-directional exchange	*(log2 p)nβ*
allgather	Partitions from all the workers are gathered on all	bucket	*pnβ*
regroup	Regroup partitions on all the workers based on the partition ID (partitions with the same ID are combined).	point-to-point direct sending	*(log2 p)nβ*
push & pull	Partitions are pushed from local tables to the global table or pulled from the global table to local tables.	point-to-point direct sending plus routing optimization	*nβ*
rotate	Build a virtual ring topology, and rotate partitions from a worker to a neighbor worker.	direct sending between neighbors on a ring	*nβ*

Harp is designed with a hybrid distributed and shared memory architecture. Figures 3 and 4 show that distributed worker nodes can be synchronized via collective communication (e.g. model rotation pipeline) while fine-grained parallelism is achieved on a shared memory architecture. Here we split the data and the model into small blocks (a row or a column of matrices), which the scheduler randomly selects while avoiding the model update conflicts on the same data block.

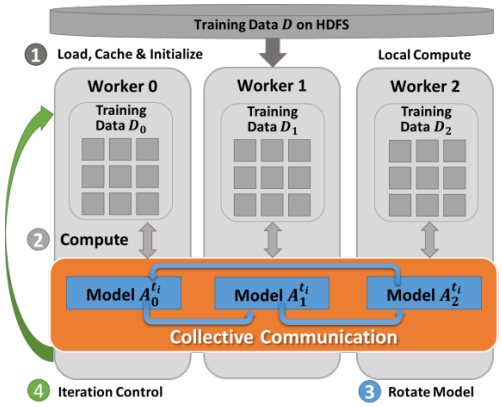


Figure 3 Model Rotation

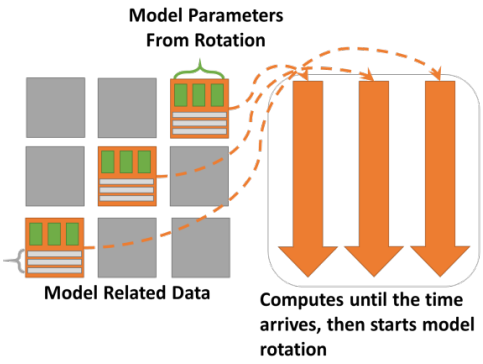


Figure 4 Dynamic Scheduler for shared memory

Hadoop/Harp on HPC

We identify the importance of the HPC-ABDS software stack [7, 8, 9, 10] illustrated by Hadoop (with the Harp plug-in) which can run K-means, Graph Layout, and Multidimensional Scaling algorithms with realistic application datasets over 4096 cores on the IU Big Red II Supercomputer (Cray/Gemini) while achieving linear speedup [8] shown in figure 5. This demonstrates the portability of HPC-ABDS to current and future (exascale) HPC systems.

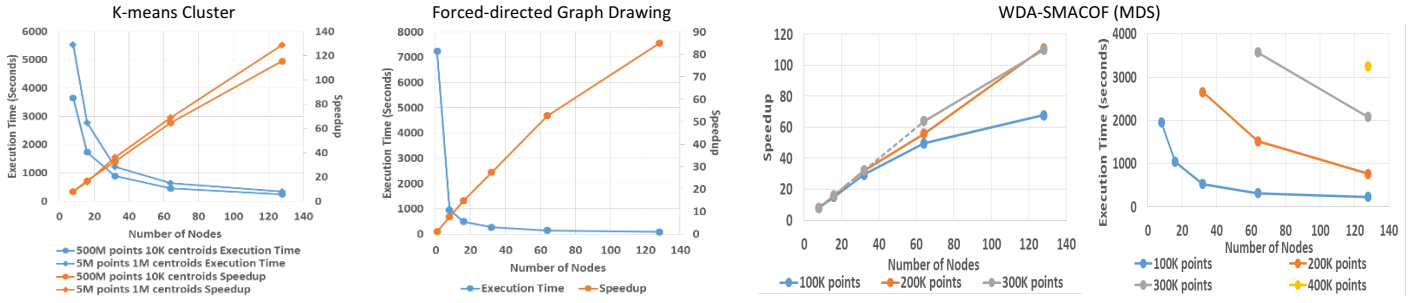


Figure 5 Performance of Hadoop/Harp Applications on the Big Red II Supercomputer

We compare in Figure 6 our Harp parallel LDA (Latent Dirichlet Allocation) [11] [12] implementation with other state-of-the-art implementations (Petuum Strads LDA [13] and Yahoo! LDA [14]) on two Intel Haswell and KNL clusters (one with Xeon E5-2699 v3 Haswell processors and another with Xeon Phi 7250F Knights Landing processors). We test all three implementations side-by-side over two large datasets. One is "clueweb" [15] (76 million documents; 1 million words, 30 billion tokens) and another is "enwiki" (3.8 million documents; 1 million words, 1 billion tokens). Both are set up with the same CGS parameters (k = 10000, α = 0.01, β = 0.01).

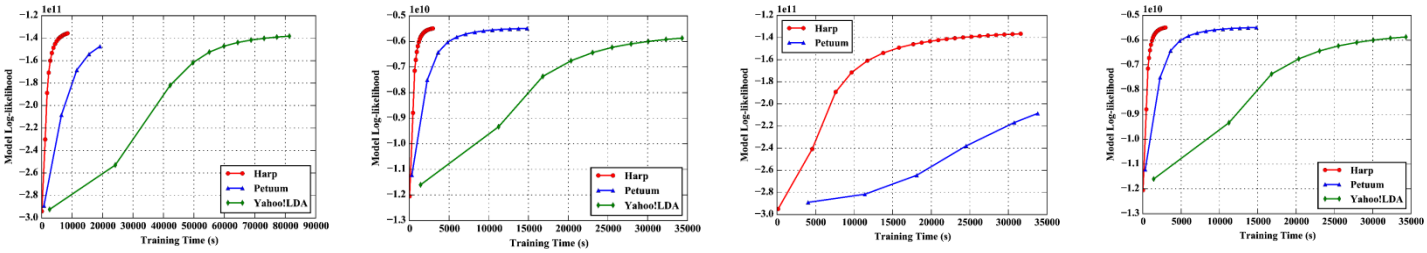


Figure 6 Performance of Hadoop/Harp LDA Applications on Intel Haswell and KNL clusters

(a). clueweb, 24x30, Haswell (b). enwiki, 10x30, Haswell (c) clueweb, 12x60, Knights Landing (d) enwiki, 5x60, Knights Landing

Figure 6 presents the model convergence in the Harp at different data and scale settings. Harp LDA implementation is significantly faster than other well known LDA implementations (2 to 4 times faster than Petuum Strads LDA and 10 times faster than Yahoo! LDA). All the implementations use the SparseLDA algorithm. Harp performance gain comes from a set of system optimizations, exploiting our understanding of the model convergence characteristics and the time complexity bounds in different stages of computation.

- applying high-performance inter-node/intra-node parallel computation models
- dynamic control on model synchronization
- loop optimization for token sampling
- utilizing efficient data structures for searching training tokens and model parameters
- caching intermediate results in the sampling process

Hadoop/Harp and DAAL

Now we introduce a new framework, Harp-DAAL and demonstrate that the combination of Big Data (e.g. Hadoop with Harp plugin) and HPC (DAAL) can achieve both productivity and performance. Harp has already been introduced and is a distributed framework based on Java implementations while DAAL is Intel's Data Analytics Accelerator Library in C++. Specifically, Hadoop/Harp invokes DAAL so as to get good performance of machine learning kernels on both Intel Haswell and KNL architectures. This way, high-level interfaces of big data tool (intro-node model synchronization) can be preserved while intra-node fine-grained parallelism is properly optimized for different HPC nodes. Such a hybrid approach can also apply to GPU and other emerging node hardware platforms.

In the experiments, we select three typical learning algorithms to evaluate our framework: 1) K-means Clustering (K-means), a computation-bounded algorithm; 2) Matrix Factorization by Stochastic Gradient Descent (MF-SGD), a computation- bounded and communication-bounded algorithm; 3) Alternating least squares (ALS), a communication-bounded algorithm. We evaluate the performance of Harp-DAAL framework and compare it with Spark and NOMAD implementations. Spark-Kmeans and Spark-ALS are pure Java applications from Apache Spark. NOMAD-SGD is a distributed MF-SGD application in C/C++ and MPI.

Through evaluating computation and communication-bounded applications, we show that Harp-DAAL combines advanced communication operations from Harp and high performance computation kernels from DAAL. Our framework achieves 15x to 40x speedups over Spark-Kmeans and 25x to 40x speedups to Spark-ALS. Compared

to NOMAD-SGD, a state-of-the- art C/C++ implementation of the MF-SGD application, we still get a factor of 2.5 improvement in performance.

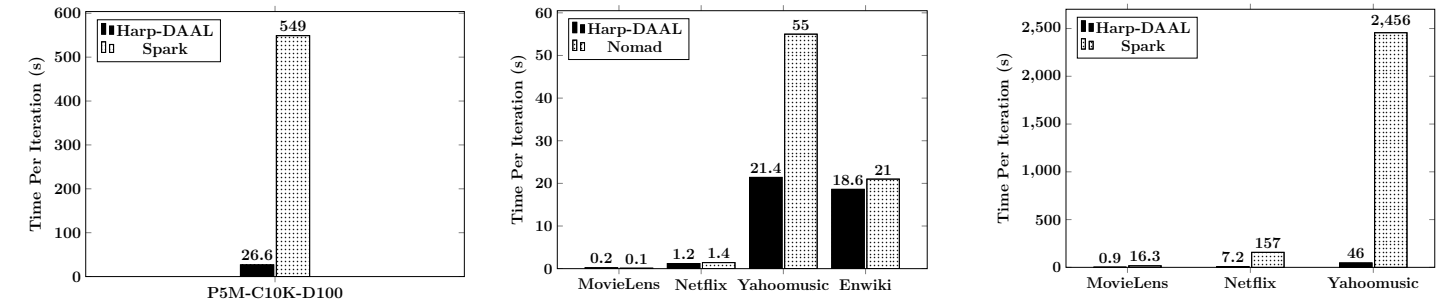


Figure 10. Performance on KNL Single Node

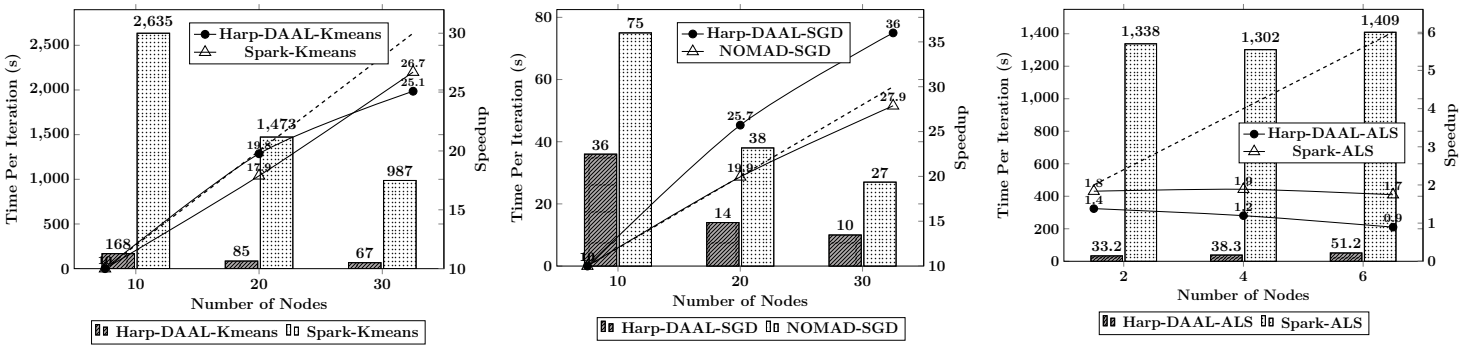


Figure 11. Performance on KNL Multi-Node

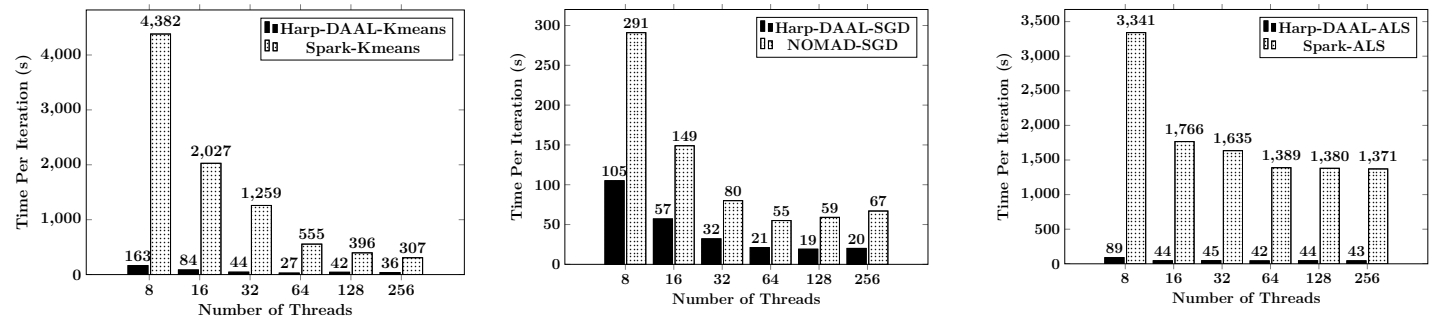


Figure 12. Breakdown of Intra-node Performance

Figure 10 shows the execution time (per iteration) on a single KNL node. Each subgraph compares two implementations on the same algorithm with different datasets. (a) K-means dataset: 5 million points, 10 thousand centroids, 1000 feature dimension; (b) MF-SGD dataset: 1) Movielens 9 million points, 40 feature dimension, 2) Netflix 100 million points, 40 feature dimension, 3) Yahoomusic 250 million points, 1000 feature dimension, 4) Enwiki 600 million points, 100 feature dimension; (c) ALS dataset: 1) Movielens 9 million points, 40 feature dimension, 2) Netflix 100 million points, 40 feature dimension, 3) Yahoomusic 250 million points, 100 feature dimension.

Figure 11 is strong scaling on multiple KNL nodes. The bars refer to the execution time of each application. Dashed line is the linear speedup of multiple nodes to a single node. Solid lines are the measured speedup of applications on multiple nodes against a single node. E.g., the value 25.7 of the marker for lines Harp-DAAL-SGD in (b) illustrates a 25.7 speedup computed as T(1)/T(20). (a) K-means dataset:20 million points, 100 thousand centroids, 100 feature dimension; (b) MF-SGD dataset, Hugewiki, with 3 billion training points, 1000 feature dimension; (c) ALS dataset, Yahoomusic, with 250 million training points, 100 feature dimension. In Figure 11 (a), Harp-DAAL-Kmeans runs 15x to 40x faster than Spark Kmeans, and shows better scalability from 10 nodes

to 20. Beyond 20 nodes, due to low computation workload on local nodes, the scalability of Harp-DAAL- Kmeans drops while Spark-Kmeans still has substantial computation workload. It suggests that Spark-Kmeans is more computation-bounded than Harp-DAAL-Kmeans because strong scalability reflects how an implementation is bounded by local computations. Since Harp-DAAL-Kmeans invokes fast MKL kernels at the low level, it is much less bounded by computation time. Figure 11 (b) shows that Harp-DAAL-SGD runs 2.5x faster than NOMAD-SGD, and it even achieves super-linear scalability on 20 nodes and 30 nodes, which is also better than the scalability of NOMAD-SGD.

Figure 12 is strong scaling on threads of a single KNL node. (a) K-means dataset: 5 million points, 10 thousand centroids, 100 feature dimension; (b) MF-SGD dataset: Yahoomusic with 250 million training points, 1000 feature dimension; (c) ALS dataset, Yahoomusic, with 250 million training points, 100 feature dimension. Figure 6 shows the performance of K- means, MF-SGD and ALS with varying numbers of threads on one single node.

Core Algorithms and Applications

Table 3: Core and Optimizations Algorithms

Algorithm	Category	Applications	Features	Computation Model	Collective Communication
K-means	Clustering	Clusters for Biology, Web, Images	Vectors	AllReduce	allreduce, regroup+allgather broadcast+reduce, push+pull
				Rotation	rotation
Multi-class Logistic Regression	Classification		Vectors, words	Regroup	
				Rotation	
				AllReduce	
Random Forests	Classification		Vectors	AllReduce	allreduce
Support Vector Machine	Classification, Regression		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	Bag of words	Rotation	rotation, allreduce
Matrix Factorization	Structure learning (Matrix completion)	Recommender system	Irregular sparse Matrix	Rotation	rotation
Multi-Dimensional Scaling	Dimension reduction	Visualization and nonlinear identification of principal components	Vectors	AllReduce	allgarther, allreduce
Subgraph Mining	Graph	Social network analysis, fraud detection, chemical informatics, bioinformatics	Graph	Rotation	rotation
Force Directed Graph Drawing	Graph	Community detection and visualization	Graph	AllReduce	allgarther, allreduce

Related Work

MapReduce is popular owing to its simplicity and scalability, yet is still slow when running iterative algorithms. Frameworks like Twister, Spark [16, 17] and HaLoop [18] solved this issue by caching the training data and developing the iterative MapReduce model. Another iterative computation model is the graph model, which abstracts data as vertices and edges and executes in Bulk Synchronous Parallel style. Pregel [19] and its open source version Giraph and Hama5 follow this design. By contrast, GraphLab [20] abstracts data as a "data graph" and uses consistency models to control vertex value updates. GraphLab (called Turi now) was later enhanced with PowerGraph [21] abstraction to reduce the communication overhead. This was also used by GraphX [22]. For all these tools, the collective communication is still implicit and coupled with the dataflow. Although some research work [23, 7, 17, 9, 24] tries to add or improve collective communication operations, they are still limited in operation types and constrained by the dataflow. The third type of iterative computation model is the Parameter Server [25] and Petuum [26] approach. Unlike the solutions above, Parameter Server does not enforce a global synchronization. Parallel workers can exchange model updates asynchronously. However, this modifies the execution dependency of the original algorithm and may affect the model convergence speed.

Table 2: Programming Models and Synchronization Patterns in Tools for Big Data Tools

Tool	Programming Model	Synchronization Pattern
MPI	a set of parallel workers are spawned with communication support between them	send/receive or collective communication operations
Hadoop	(iterative) MapReduce, DAG-like distributed job execution flow may be supported	disk-based shuffle between Map stage and Reduce stage
Twister		in-memory regroup between Map stage and Reduce stage, "broadcast" and "aggregate"
Spark		RDD transformations on RDD, "broadcast" and "aggregate"
Giraph	BSP model, data are expressed as vertices and edges in a graph	graph-based message communication following Pregel model (messages are sent between neighbor vertices)
Hama		graph-based communication following Pregel model or direct message communication between workers.
GraphLab (Turi)		graph-based communication through caching and fetching of ghost vertices and edges, or the communication between master vertex and its replicas in Power-Graph (GAS) model
GraphX		graph-based communication supports both Pregel model and PowerGraph model
Parameter Server	BSP model, or loosely synchronized on the parameter server	asynchronous "push" and "pull" calls are used for fetching model parameters from parameter servers to workers
Petuum		in addition to asynchronous "push" and "pull" calls, the framework allows scheduling model parameters between workers

In a broader context, the schematics in Figure 13 illustrate how our research enables us to classify data intensive computation into six computation models that map into six distinct system architectures. The central batch architectures are categories 1 to 4, which correspond exactly to the four forms of MapReduce. Category 4 is the classic distributed memory model, category 5 is the new Map-streaming model we pioneered with the IU network science group [27], and category 6 is the shared memory architecture needed for some graph algorithms as well as large memory applications.

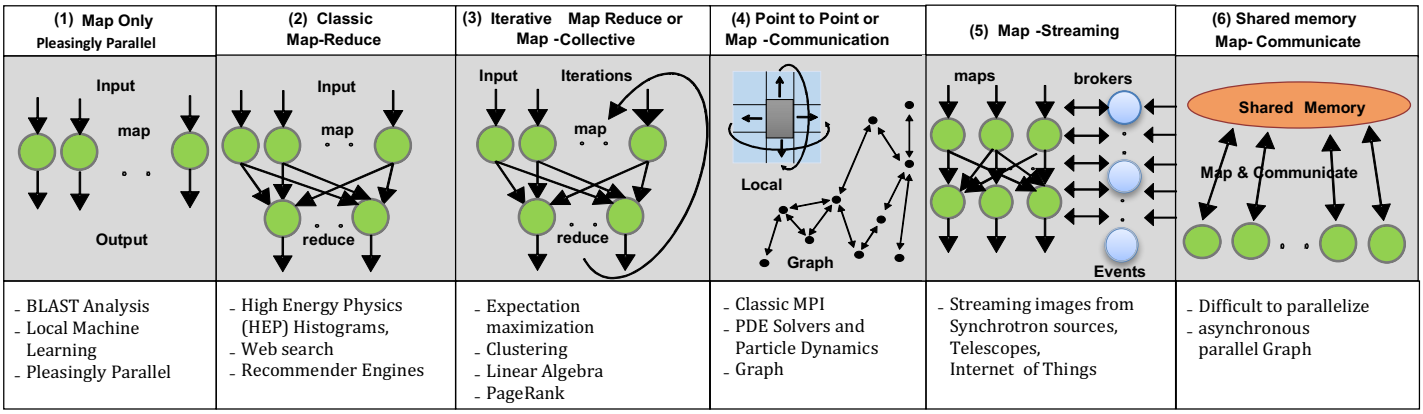
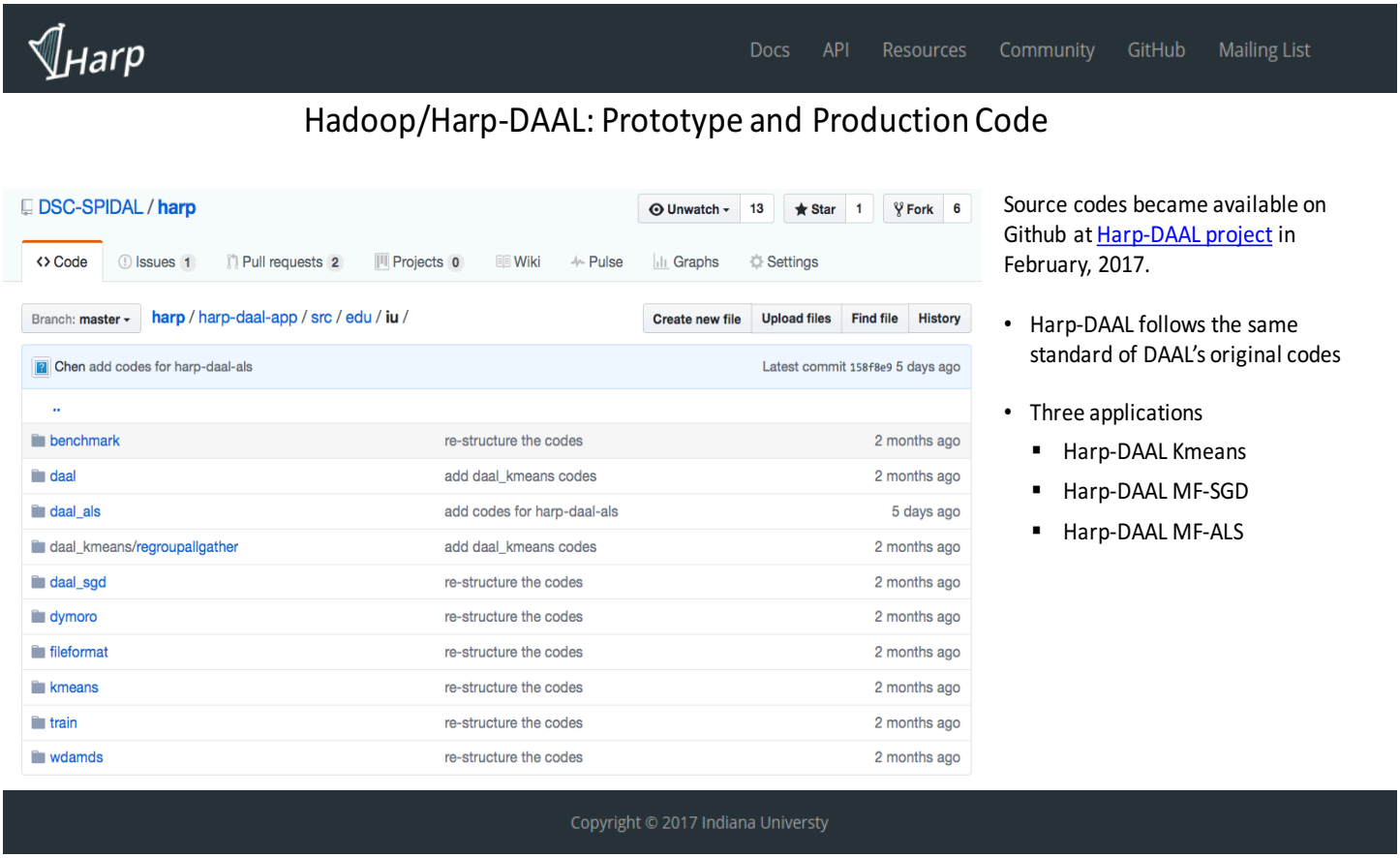


Figure 13 Six Computation Models for Data Analytics

Map-Collective and Map-(Point to Point) Communication are separated following the Apache projects Hadoop, Spark, and Giraph which focus on these cases. These programming models or runtimes differ in communication style (bandwidth versus latency), application abstraction (key-value versus graph), scheduling and load-balancing. HPC with MPI suggests that one could integrate categories 3 and 4 into a single environment. This approach is illustrated by the Harp plug-in which supports both models [28]. Most stream processing engines such as Apache Storm organize distributed workers in the form of a directed acyclic graph, which makes it a challenge to synchronize the state of parallel workers dynamically.

Conclusions and Future Work



In summary, Harp is designed to use collective communication techniques to improve the performance of synchronization in parallel iterative algorithms. The advantage of using collective communication is that the performance of many synchronization patterns can be optimized. We categorize four types of computation models (locking, rotation, allreduce, asynchronous) based on the synchronization patterns and the effectiveness of the model parameter update. These are implemented with Harp collective communication interfaces in order to simplify the implementation of parallel iterative algorithms. These enhancements are implemented as a plugin to Hadoop so that Harp can support machine learning and data analysis with HPC capabilities for the community ecosystems. Harp is portable to both Cloud (Hadoop) and HPC platforms (e.g. Big Red II supercomputing, Intel Haswell and Knights Landing clusters).

The Harp framework has been released as open source project that is available at public github domain. It has been used by over 350 IU students for the past two years. It has a collection of iterative machine learning and data analysis algorithms that has been tested and benchmarked on Cloud and HPC platforms including Haswell and Knights Landing hardware.

References

[1] Judy Qiu, Shantenu Jha, Andre Luckow, Geoffrey C. Fox, Towards HPC-ABDS: An Initial High-Performance Big Data Stack, accepted to the proceedings of ACM 1st Big Data Interoperability Framework Workshop: Building Robust Big Data ecosystem, NIST special publication, March 13-21, 2014.

[2] NSF Datanet: CIF21 DIBBs: Middleware and High Performance Analytics Libraries for Scalable Data Science Progress Report on July, 2016.

[3] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, G. Fox, "Twister: A Runtime for Iterative MapReduce", in Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference, Chicago, Illinois, June 20-25, 2010.

[4] B. Zhang, Y. Ruan, and J. Qiu, "Harp: Collective Communication on Hadoop," in IC2E, 2015.

[5] R. Gemulla, P. Haas, E. Nijkamp and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," SIGKDD, 2011.

[6] H.-F. Yu, C.-J. Hsieh, S. Si and I. Dhillon. "Scalable coordinate descent approaches to parallel matrix factorization for recommender systems," ICDM 2012.

[7] B. Zhang and J. Qiu, "High performance clustering of social images in a map-collective programming model," in Proceedings of the 4th annual Symposium on Cloud Computing, 2013.

[8] B. Zhang, B. Peng and J. Qiu, "High Performance LDA through Collective Model Communication Optimization", International Conference on Computational Science (ICCS), June 6-8, 2016.

[9] Thilina Gunarathne, Judy Qiu, Dennis Gannon, Towards a Collective Layer in the Big Data Stack, Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing Conference (IEEE/ACM CCGrid 2014).

[10] S. Wu, A. Koppula, J. Qiu, Integrating Pig with Harp to Support Iterative Applications with Fast Cache and Collective Communication, Proceedings of the 5th International Workshop on Data Intensive Computing in the Cloud (DataCloud) at SC14, November 21, 2014.

[11] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. The Journal of Machine Learning Research, 3:993–102, 2003.

[12] P. Resnik and E. Hardist. Gibbs sampling for the uninitiated. Technical report, University of Maryland, 2010.

[13] Petuum LDA. https://github.com/petuum/bosen/wi ki/Latent-Dirichlet-Allocation.

[14] Yahoo! LDA. https://github.com/sudar/Yahoo LDA.

[15] clueweb. http://lemurproject.org/clueweb09/

[16] M. Zaharia et al., "Spark: cluster computing with working sets," in Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, vol. 10, 2010, p. 10.

[17] M. Chowdhury et al., "Managing data transfers in computer clusters with orchestra," ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, pp. 98–109, 2011.

[18] Y. Bu et al., "Haloop: efficient iterative data processing on large clusters," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 285–296, 2010.

[19] G Malewicz, MH Austern, AJC Bik, JC Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10), Pages 135-146, Indianapolis, June 6-10, 2010.

[20] Y. Low et al., "Distributed graphlab: a framework for machine learning and data mining in the cloud," Proceedings of the VLDB Endowment, vol. 5, no. 8, pp. 716–727, 2012.

[21] J. E. Gonzalez et al., "PowerGraph: distributed graph-parallel computation on natural graphs," in OSDI, vol. 12, 2012, p. 2.

[22] R. Xin et al., "Graphx: A resilient distributed graph system on spark," in First International Workshop on Graph Data Management Experiences and Systems, 2013, p. 2.

[23] J. Qiu and B. Zhang, "Mammoth data in the cloud: clustering social images," in Clouds, Grids and Big Data, ser. Advances in Parallel Computing. IOS Press, 2013. 63

[24] T. Gunarathne, J. Qiu, and D. Gannon, "Towards a collective layer in the big data stack," in Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, 2014, pp. 236–245.

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

[26] E. P. Xing et al., "Petuum: A New Platform for Distributed Machine Learning on Big Data," IEEE Transactions on Big Data, 2015.

[27] B. Zhang, B. Peng and J. Qiu, "Model-Centric Computation Abstractions in Machine Learning Applications", BeyondMR workshop with SIGMOD, July 1, 2016

[28] Shantenu Jha, Judy Qiu, Andre Luckow, Pradeep Mantha, Geoffrey C.Fox, A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures, Proceedings of the 3rd International Congress on Big Data Conference (IEEE BigData 2014).

[29] The code and documentation of Harp framework can be found at https://dsc-spidal.github.io/harp/