A Collective Communication Layer for the Software Stack of Big Data Analytics (Thesis Proposal)

Bingjing Zhang

School of Informatics and Computing
Indiana University Bloomington

December 8, 2015

Table of Contents

Table of Contents

Big Data Analytics

What is "big data" in analytics?

- Big for huge input data
- Big for huge intermediate data

Application examples - machine learning

- Widely used in computer vision, text mining, advertising, recommender systems, network analysis, and genetics
- Training data (input) & model data (intermediate)

Scaling up these applications is difficult for systems!

- For training data - use caching
- For model data - limited support for model synchronization

Machine Learning & Collective Communication

Model synchronization in machine learning

- Fine-grained control - what, when, where, how
- High communication overhead
- Performed iteratively

Suggest using collective communication abstractions!

- Serve different communication patterns
- Routing optimization

The System Solution to Big Data Problems

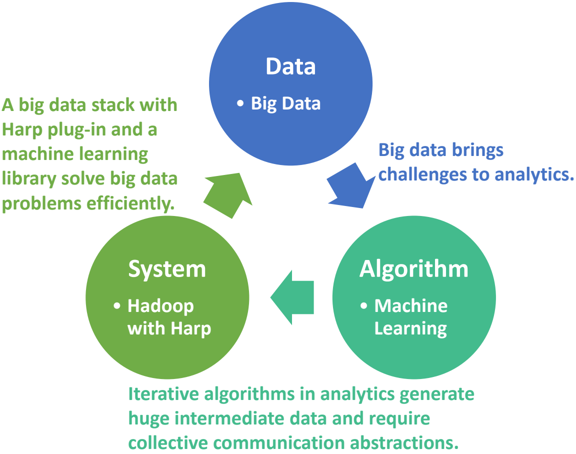
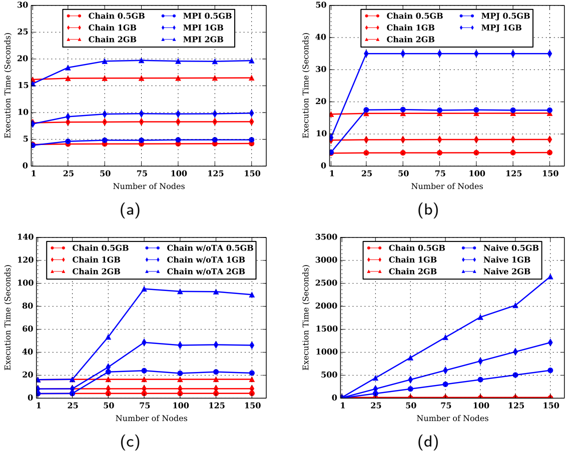


Table of Contents

Contemporary Big Data Tools

Tool	Computation Model	Data Abstraction	Communication Pattern
MPI [1]	Loosely Synchronous	N/A	Arrays and objects sending/receiving or collective communication operations
Hadoop [2]	(Iterative) MapReduce	Key-Values	Shuffle (disk-based) between Map stage and Reduce stage
Twister [3]			Regroup (in-memory) between Map stage and Reduce stage, "broadcast" and "aggregate"
Spark [4]		RDD	RDD Transformations on RDD, "broadcast" and "aggregate"
Dryad [5]	DAG	N/A	Communication is between two connected vertex processes in the execution of DAG
Giraph [6]	Graph/BSP	Graph	Graph-based message communication following Pregel model
Hama [7]			Graph-based message communication following Pregel model or direct message communication between workers
GraphLab (Dato) [8, 9, 10]			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 [11]			Graph-based communication supports Pregel model and PowerGraph model

An Example of Chain Broadcast



Performance comparison between "broadcast" methods: (a) Chain vs. MPI (b) Chain vs. MPJ (c) Chain vs. Chain without topology-awareness (d) Chain vs. Naive method

Table of Contents

Research Challenges

Unite collective communication abstractions from different tools

- Each tool has its own computation model, data and communication abstractions
- Provide a horizontally abstracted collective communication layer

Optimize collective communication operations

- Naive implementation could harm the performance
- Optimized implementation

Match collective communication to machine learning applications

- Each machine learning application has its own features of model synchronization
- Find suitable operations or provide suitable abstractions

Table of Contents

Contributions

- **A collective communication abstraction layer**
  - with data abstractions and communication abstractions
- **A MapCollective programming model**
  - on top of the communication abstraction layer
  - allows users to invoke collective communication operations to synchronize parallel workers.
- **A communication library**
  - Hadoop plug-in

The Concept of Harp Plug-in

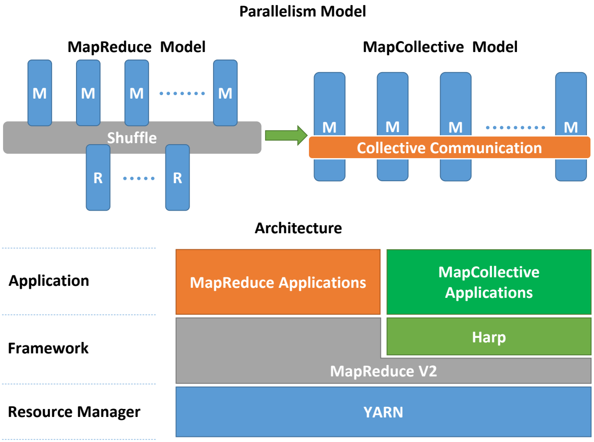
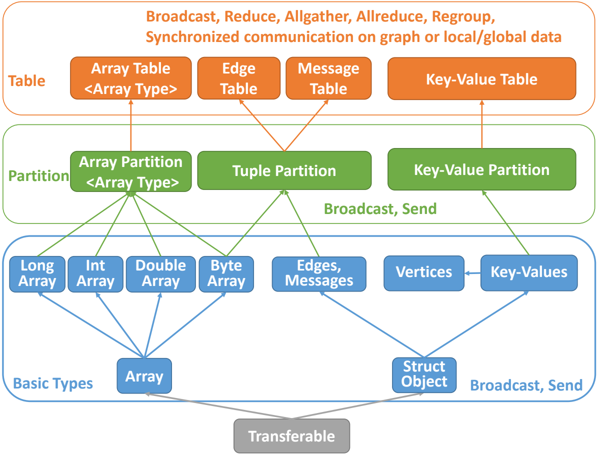


Table of Contents

Hierarchical Data Abstractions

- $\rightarrow$ as arrays, key-values, or edges and messages in graphs
- $\uparrow$ from basic types to partitions and tables



Collective Communication Operations

- **Collective communication adapted from MPI operations [12]**
  - "broadcast"
  - "reduce"
  - "allgather"
  - "allreduce"
- **Collective communication derived from MapReduce "shuffle-reduce" operation**
  - "regroup" operation with "combine & reduce" support
- **Collective communication based on graph**
  - "send messages to vertices"
- **Collective communication abstracted from data parallelism and model parallelism in machine learning applications**
  - data parallelism through "syncLocalWithGlobal" and "syncGlobalWithLocal"
  - model parallelism through "rotateGlobal"

Collective Communication Operations (cont'd)

Operation	Algorithm	Time Complexity
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-allgather	$2n\beta$
regroup	point-to-point direct sending	$n\beta$
send messages to vertices	point-to-point direct sending	$n\beta$
syncLocalWithGlobal	point-to-point direct sending plus routing optimization	$pn\beta$
syncGlobalWithLocal	point-to-point direct sending plus routing optimization	$n\beta$
rotateGlobal	direct sending between neighbors	$n\beta$

Note in Column "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 α is neglected and the time complexity of the "point-to-point direct sending" algorithm is estimated regardless of potential network conflicts.

MapCollective Programming Model

- **BSP style**
  - each worker is deployed on a compute node
- **Separate inter-node parallelism and intra-node parallelism**
  - This is a world of “big” machines!
  - inter-node
    - use collective communication to synchronize parallel workers
  - intra-node
    - parallel threads with running state control

Table of Contents

Layered Architecture

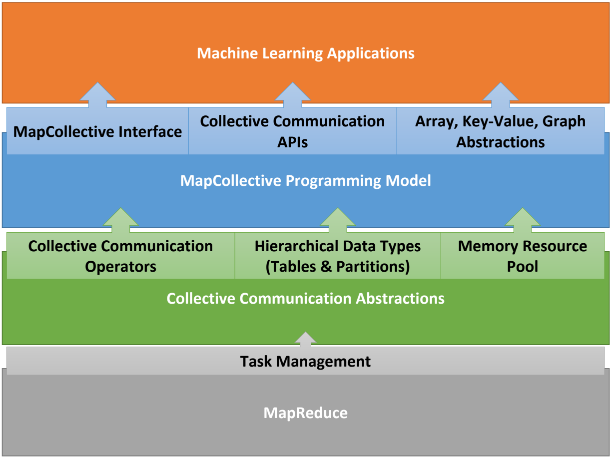


Table of Contents

Machine Learning Applications Implemented in Harp

Application	Model Size	Model Dependency	Parallelism	Communication
K-means Clustering [13]	Usually in MB level, but can grow to GB level	All	Data Parallelism	allreduce
WDA-SMACOF [14]	A few MBs	All	Data Parallelism	allgather & allreduce
LDA [15]	From a few GBs to 10s of GBs, or even larger	Partial	Data Parallelism	syncGlobalWithLocal & syncLocalWithGlobal
			Model Parallelism	rotateGlobal

Note: "model dependency" refers to the model data requirement in each local computation. "all" means the local computation needs all the model data. "partial" means local computation may not need all the model data. In "parallelism", "Data Parallelism" means only the training data are split among parallel workers, and each worker computes on a local model and updates it through the global model synchronization with other workers. "Model Parallelism" means in addition to splitting the training data over parallel workers, the global model data is split between parallel workers and rotated during computation.

Table of Contents

K-means Clustering

- **Clustering 500 million 3D points into 10 thousand clusters**
  - The input data is about 12GB
  - The ratio of points to clusters is 50000:1
- **Clustering 5 million 3D points into 1 million clusters**
  - The input data size is about 120MB
  - The ratio of points to clusters is 5:1

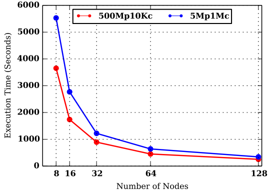
WDA-SMACOF

- **SMACOF (Scaling by MAjorizing a COmplicated Function)**
  - minimizes the difference between distances from points in the original space and their distances in the new space through iterative stress majorization
- **WDA-SMACOF is an improved version of the original SMACOF**
  - deterministic annealing
  - conjugate gradient
  - nested iterations
  - "allgather" and "allreduce"
- **Runs with 100K, 200K, 300K and 400K points**
  - each point represents a gene sequence [16]
  - 100K - 140GB
  - 200K - 560GB
  - 300K - 1.3TB
  - 400K - 2.2TB

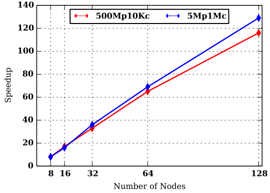
Test Environment

- **Big Red II [17]**
  - "cpu" queue
  - maximum number of nodes per job submission - 128
  - each node has 32 threads and 64GB memory
  - Cluster Compatibility Mode
  - connected with Cray Gemini interconnect

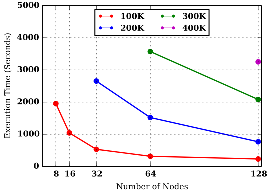
Performance Results



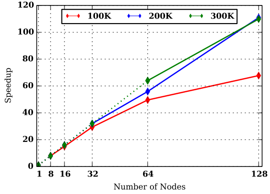
(a)



(b)



(c)



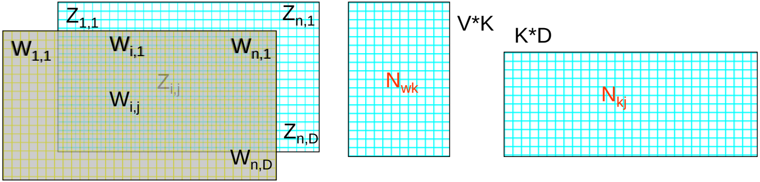
(d)

(a) Execution time of k-means (b) Speedup of k-means (c) Execution time of WDA-SMACOF (d) Speedup of WDA-SMACOF

Table of Contents

Gibbs Sampling in LDA

- Observed data: W_{ij}, word on position i in doc j
- Try to estimate the latent variables (Model Data)
  - Z_{ij}, topic assignment accordingly to W_{ij}
  - N_{wk}, count matrix for word-topic distribution
  - N_{kj}, count matrix for topic-document distribution
- With parameters
  - Concentration Parameters - α, β, control model sparseness
  - D documents, V vocabulary size, K topics



Gibbs Sampling in LDA (cont'd)

Initialize:

sample topic index $z_{ij} = k \sim Mult(1/K)$

Repeat until converge:

for all documents $j \in [1, D]$ **do**

for all words position $i \in [1, N_m]$ in document j **do**

// for the current assignment k to a token t of word w_{ij}, decrease counts

$n_{kj} -= 1; n_{tk} -= 1;$

// multinomial sampling

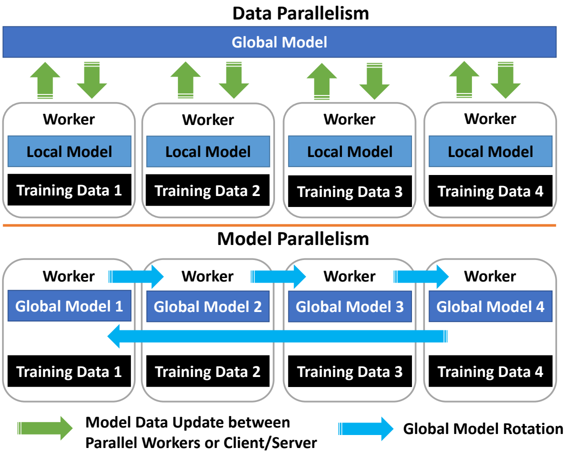
sample new topic index

$$k' \sim p(z_{ij}|z^{\neg ij}, w) \propto \frac{N_{wk}^{\neg ij} + \beta}{\sum_w N_{wk}^{\neg ij} + V\beta} \left(N_{kj}^{\neg ij} + \alpha\right)$$

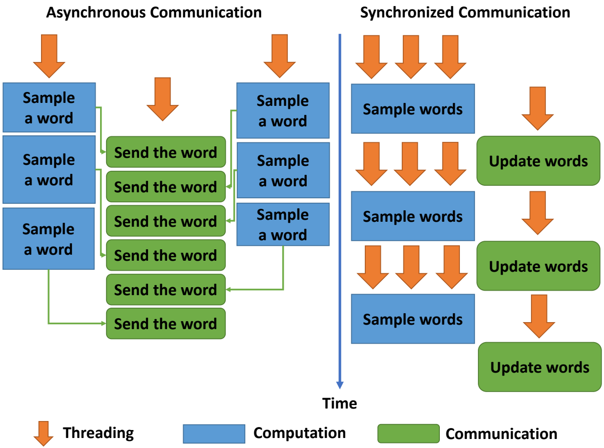
// for the new assignment k' to the token t of word w_{ij}, increase counts

$n_{k'j} += 1; n_{tk'} += 1;$

Data Parallelism vs. Model Parallelism in LDA



Synchronized Method vs. Asynchronous Method in LDA

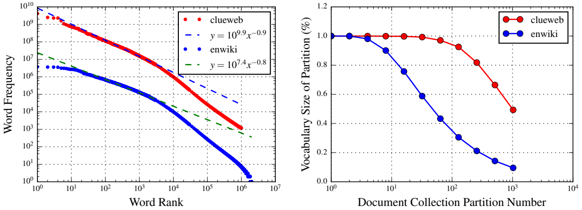


LDA Work Using CGS Algorithm

Application	Algorithm	Parallelism	Communication
PLDA [18]	CGS [19] (sample by docs)	D. P.	allreduce (sync)
Dato [20]	CGS (sample by doc-word edge)	D. P.	GAS (sync)
Yahoo! LDA [21, 22]	CGS (SparseLDA [23] & sample by docs)	D. P.	client-server (async)
Peacock [24]	CGS (SparseLDA & sample by words)	D. P. (M. P. in local)	client-server (async)
Parameter Server [25]	CGS (combined with other methods)	D. P.	client-server (async)
Petuum 0.93 [26]	CGS (SparseLDA & sample by docs)	D. P.	client-server (async)
Petuum 1.1 [27, 28]	CGS (SparseLDA & sample by words)	M. P. (include D. P.)	ring/star topology (async)

Note: "D. P." refers to Data Parallelism. "M. P." refers to Model Parallelism.

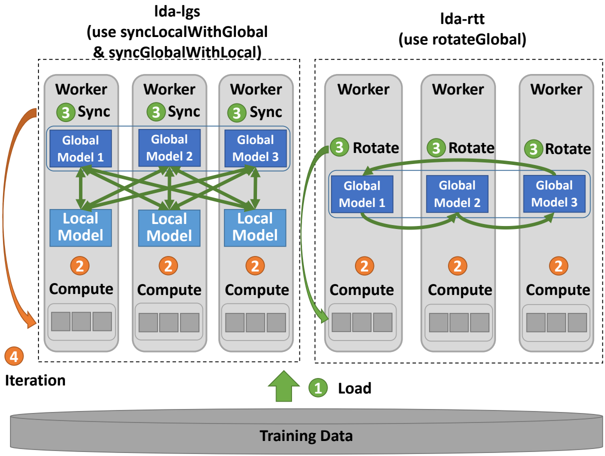
Power-law Distribution



(a) (b)

(a) Zipf's Law of the word frequency (b) Number of words per partition under different partitioning

LDA Implementations



Test Environment in LDA Experiments

- **Juliet Intel Haswell cluster [29]**
  - 32 nodes each with two 18-core 36-thread Xeon E5-2699 processors and 96 nodes each with two 12-core 24-thread Xeon E5-2670 processors.
  - 128GB memory
  - network - 1Gbps Ethernet (eth) and Infiniband (ib)
- **In LDA experiments...**
  - 31 nodes with Xeon E5-2699 and 69 nodes with Xeon E5-2670 are used to form a cluster of 100 nodes with 40 threads
  - use ib in default

Training Datasets Used In LDA Experiments

- **The total number of model parameters is kept as 10 billion on all the datasets.**

Dataset	enwiki	clueweb	bi-gram	gutenberg
Num. of Docs	3.8M	50.5M	3.9M	26.2K
Num. of Tokens	1.1B	12.4B	1.7B	836.8M
Vocabulary	1M	1M	20M	1M
Doc Len. AVG/STD	293/523	224/352	434/776	31879/42147
Highest Word Freq.	1714722	3989024	459631	1815049
Lowest Word Freq.	7	285	6	2
Num. of Topics	10K	10K	500	10K
Init. Model Size	2.0GB	14.7GB	5.9GB	1.7GB

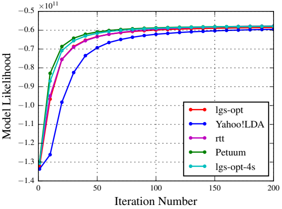
Note: Both "enwiki" and "bi-gram" are English articles from Wikipedia [30]. "clueweb" is a 10% dataset from ClueWeb09, which is a collection of English web pages [31]. "gutenberg" is comprised of English books from Project Gutenberg [32].

Implementations Used In LDA Experiments

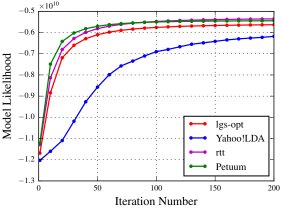
DATA PARALLELISM	
lgs	- "lda-lgs" impl. with no routing optimization - Slower than "lgs-opt"
lgs-opt	- "lgs" with routing optimization - Faster than Yahoo! LDA on "enwiki" with higher model likelihood
lgs-opt-4s	- "lgs-opt" with 4 rounds of model synchronization per iteration; each round uses 1/4 of the training data - Performance comparable to Yahoo! LDA on "clueweb" with higher model likelihood
Yahoo! LDA	- Master branch on GitHub [33]
MODEL PARALLELISM	
rtt	- "lda-rtt" impl. - Speed comparable with Petuum on "clueweb" but 3.9 times faster on "bi-gram" and 5.4 times faster on "gutenberg"
Petuum	- Version 1.1 [34]

Note: Proposed implementations are indicated in bold.

LDA Model Convergence Speed Per Iteration



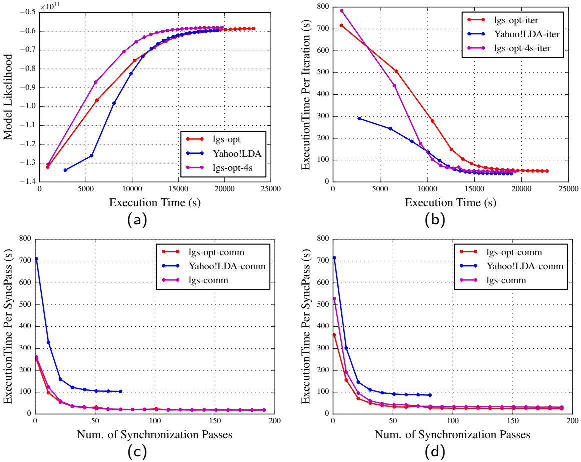
(a)



(b)

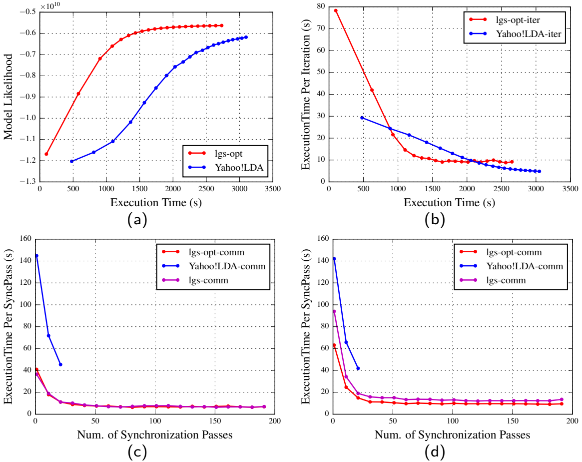
(a) Model convergence speed of "clueweb" on iterations (b) Model convergence speed of "enwiki" on iterations

LDA Data Parallelism on “clueweb”



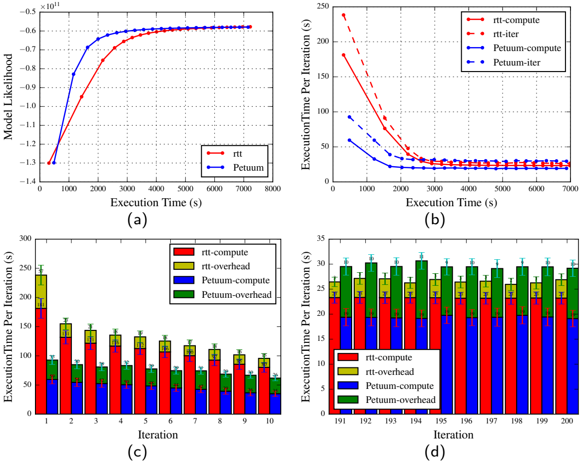
(a) Elapsed Execution Time vs. Model Likelihood (b) Elapsed Execution Time vs. Iteration Execution Time (c) Num. of Sync. Passes vs. Sync. Time per Pass with ib (d) Num. of Sync. Passes vs. Sync. Time per Pass with eth

LDA Data Parallelism on “enwiki”



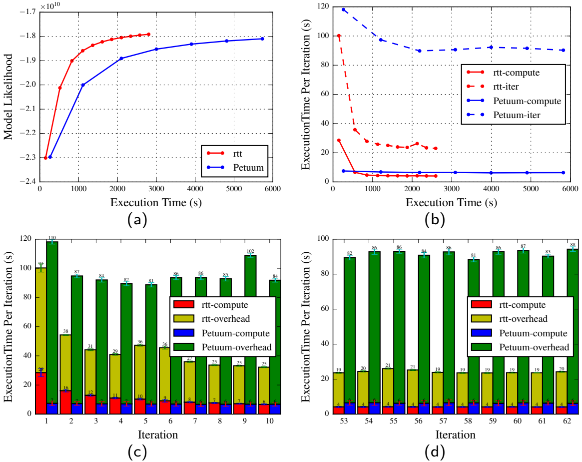
(a) Elapsed Execution Time vs. Model Likelihood (b) Elapsed Execution Time vs. Iteration Execution Time (c) Num. of Sync. Passes vs. Sync. Time per Pass with ib (d) Num. of Sync. Passes vs. Sync. Time per Pass with eth

LDA Model Parallelism on “clueweb”



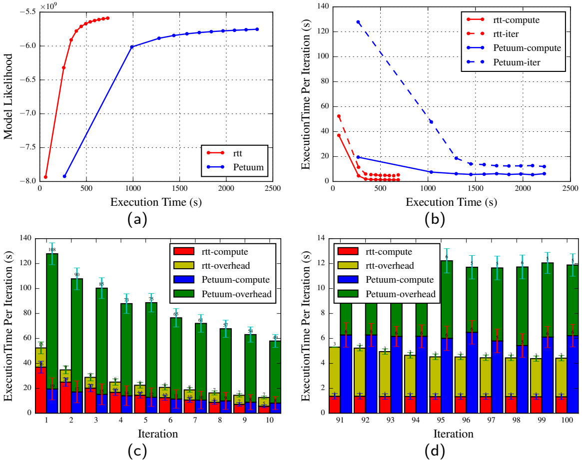
(a) Elapsed Execution Time vs. Model Likelihood (b) Elapsed Execution Time vs. Iteration Execution Time (c) First 10 Iteration Execution Times (d) Final 10 Iteration Execution Times

LDA Model Parallelism on “bi-gram”



(a) Elapsed Execution Time vs. Model Likelihood (b) Elapsed Execution Time vs. Iteration Execution Time (c) First 10 Iteration Execution Times (d) Final 10 Iteration Execution Times

LDA Model Parallelism on “gutenburg”



(a) Elapsed Execution Time vs. Model Likelihood (b) Elapsed Execution Time vs. Iteration Execution Time (c) First 10 Iteration Execution Times (d) Final 10 Iteration Execution Times

Table of Contents

Conclusion

- Collective communication is essential to the performance of model synchronization in the machine learning applications.
- The research on LDA shows that improving the efficiency of model synchronization allows the model to converge faster, shrink the model size, and further reduce the later computation time.
- In future work, it is expected to improve the performance of other machine learning applications through applying the collective communication abstraction on the model synchronization.

References I

[1] D. W. Walker and J. J. Dongarra, "MPI: a standard message passing interface," in *Supercomputer*, vol. 12, 1996, pp. 56–68.
[2] "Hadoop," http://hadoop.apache.org.
[3] J. Ekanayake *et al.*, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 810–818.
[4] 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.
[5] M. Isard *et al.*, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, 2007, pp. 59–72.
[6] "Giraph," https://giraph.apache.org.
[7] "Hama," https://hama.apache.org.
[8] 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.
[9] J. E. Gonzalez *et al.*, "PowerGraph: distributed graph-parallel computation on natural graphs," in *OSDI*, vol. 12, 2012, p. 2.
[10] "Dato," https://dato.com.
[11] 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.
[12] E. Chan *et al.*, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.
[13] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.
[14] Y. Ruan and G. Fox, "A robust and scalable solution for interpolative multidimensional scaling with weighting," in *IEEE 9th International Conference on eScience*, 2013, pp. 61–69.
[15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–102, 2003.
[16] Y. Ruan *et al.*, "Integration of clustering and multidimensional scaling to determine phylogenetic trees as spherical phylograms visualized in 3 dimensions," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, 2014, pp. 720–729.

References II

[17] "Big Red II," https://kb.iu.edu/data/bcqt.html.

[18] Y. Wang *et al.*, "PLDA: parallel latent dirichlet allocation for large-scale applications," *Algorithmic Aspects in Information and Management*, pp. 301–314, 2009.

[19] P. Resnik and E. Hardist, "Gibbs sampling for the uninitiated," University of Maryland, Tech. Rep., 2010.

[20] "Dato LDA," https://github.com/dato-code/PowerGraph/blob/master/toolkits/topic_modeling/topic_modeling.dox.

[21] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," in *VLDB*, vol. 3, no. 1-2, 2010, pp. 703–710.

[22] A. Ahmed *et al.*, "Scalable inference in latent variable models," in *WSDM*, 2012, pp. 123–132.

[23] L. Yao, D. Mimno, and A. McCallum, "Efficient methods for topic model inference on streaming document collections," in *KDD*, 2009, pp. 937–946.

[24] Y. Wang *et al.*, ""Peacock: learning long-tail topic features for industrial applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, no. 4, 2015.

[25] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014, pp. 583–598.

[26] Q. Ho *et al.*, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[27] S. Lee *et al.*, "On model parallelization and scheduling strategies for distributed machine learning," in *NIPS*, 2014, pp. 2834–2842.

[28] E. P. Xing *et al.*, "Petuum: a new platform for distributed machine learning on big data," in *KDD*, 2013.

[29] "FutureSystems," https://portal.futuresystems.org.

[30] "wikipedia," https://www.wikipedia.org.

[31] "clueweb," http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php?page=Dataset+Information.

[32] "gutenburg," https://www.gutenberg.org.

[33] "Yahoo! LDA," https://github.com/sudar/Yahoo_LDA.

[34] "Petuum LDA," https://github.com/petuum/bosen/wiki/Latent-Dirichlet-Allocation.

List of Publications

1. **B. Zhang**, Y. Ruan, and J. Qiu, "Harp: Collective communication on hadoop," in *Proceedings of IEEE International Conference on Cloud Engineering (IC2E)*, 2015.
2. **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.
3. 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.
4. T. Gunarathne, **B. Zhang**, T.-L. Wu, and J. Qiu, "Scalable parallel computing on clouds using twister4azure iterative mapreduce," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1035–1048, 2013.
5. T. Gunarathne, **B. Zhang**, T.-L. Wu, and J. Qiu, "Portable parallel programming on cloud and hpc: Scientific applications of twister4azure," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 2011, pp. 97–104.
6. **B. Zhang**, Y. Ruan, T.-L. Wu, J. Qiu, A. Hughes, and G. Fox, "Applying twister to scientific applications," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 25–32.
7. J. Qiu, J. Ekanayake, T. Gunarathne, J. Y. Choi, S.-H. Bae, H. Li, **B. Zhang**, T.-L. Wu, Y. Ruan, and S. Ekanayake, "Hybrid cloud and cluster computing paradigms for life science applications," *BMC bioinformatics*, vol. 11, 2010.
8. J. Ekanayake, H. Li, **B. Zhang**, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 810–818.