Table of Contents

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
   - Collective Communication Abstractions
   - Layered Architecture
5. Machine Learning Library
   - K-means Clustering & WDA-SMACOF
   - LDA
6. Conclusion
# Table of Contents

1. **Research Backgrounds**

2. **Related Work**

3. **Research Challenges**

4. **Harp**

5. **Machine Learning Library**

6. **Conclusion**
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

A big data stack with Harp plug-in and a machine learning library solve big data problems efficiently.

Iterative algorithms in analytics generate huge intermediate data and require collective communication abstractions.

Big data brings challenges to analytics.
## Table of Contents

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
5. Machine Learning Library
6. Conclusion
## Contemporary Big Data Tools

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

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
5. Machine Learning Library
6. Conclusion
A Collective Communication Layer

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

1 Research Backgrounds
2 Related Work
3 Research Challenges
4 Harp
5 Machine Learning Library
6 Conclusion
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

1 Research Backgrounds
2 Related Work
3 Research Challenges
4 Harp
   - Collective Communication Abstractions
   - Layered Architecture
5 Machine Learning Library
   - K-means Clustering & WDA-SMACOF
   - LDA
6 Conclusion
Hierarchical Data Abstractions

- → as arrays, key-values, or edges and messages in graphs
- ↑ from basic types to partitions and tables

Diagram:
- Tables
  - Array Table
  - Edge Table
  - Message Table
  - Key-Value Table
- Partitions
  - Array Partition
  - Tuple Partition
  - Key-Value Partition
- Basic Types
  - Long Array
  - Int Array
  - Double Array
  - Byte Array
  - Edges, Messages
  - Vertices
- Struct Object
- Transferable

Broadcast, Reduce, Allgather, Allreduce, Regroup, Synchronized communication on graph or local/global data
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)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Algorithm</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>broadcast</td>
<td>chain</td>
<td>(n\beta)</td>
</tr>
<tr>
<td></td>
<td>minimum spanning tree</td>
<td>((\log_2 p)n\beta)</td>
</tr>
<tr>
<td>reduce</td>
<td>minimum spanning tree</td>
<td>((\log_2 p)n\beta)</td>
</tr>
<tr>
<td>allgather</td>
<td>bucket</td>
<td>(pn\beta)</td>
</tr>
<tr>
<td>allreduce</td>
<td>bi-directional exchange</td>
<td>((\log_2 p)n\beta)</td>
</tr>
<tr>
<td></td>
<td>regroup-allgather</td>
<td>(2n\beta)</td>
</tr>
<tr>
<td>regroup</td>
<td>point-to-point direct sending</td>
<td>(n\beta)</td>
</tr>
<tr>
<td>send messages to vertices</td>
<td>point-to-point direct sending</td>
<td>(n\beta)</td>
</tr>
<tr>
<td>syncLocalWithGlobal</td>
<td>point-to-point direct sending</td>
<td>(pn\beta)</td>
</tr>
<tr>
<td></td>
<td>plus routing optimization</td>
<td>(pn\beta)</td>
</tr>
<tr>
<td>syncGlobalWithLocal</td>
<td>point-to-point direct sending</td>
<td>(n\beta)</td>
</tr>
<tr>
<td></td>
<td>plus routing optimization</td>
<td>(n\beta)</td>
</tr>
<tr>
<td>rotateGlobal</td>
<td>direct sending between</td>
<td>(n\beta)</td>
</tr>
<tr>
<td></td>
<td>neighbors</td>
<td></td>
</tr>
</tbody>
</table>

Note in Column “Time Complexity”, \(p\) is the number of processes, \(n\) is the number of input data items per worker, \(\beta\) is the per data item transmission time, communication startup time \(\alpha\) 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

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
   - Collective Communication Abstractions
   - Layered Architecture
5. Machine Learning Library
   - K-means Clustering & WDA-SMACOF
   - LDA
6. Conclusion
Layered Architecture

- Machine Learning Applications
- MapCollective Interface
- Collective Communication APIs
- Array, Key-Value, Graph Abstractions
- MapCollective Programming Model
- Collective Communication Operators
- Hierarchical Data Types (Tables & Partitions)
- Memory Resource Pool
- Collective Communication Abstractions
- Task Management
- MapReduce
Table of Contents

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
5. Machine Learning Library
6. Conclusion
## Machine Learning Applications Implemented in Harp

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

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

1 Research Backgrounds
2 Related Work
3 Research Challenges
4 Harp
   • Collective Communication Abstractions
   • Layered Architecture
5 Machine Learning Library
   • K-means Clustering & WDA-SMACOF
   • LDA
6 Conclusion
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) Execution time of k-means (b) Speedup of k-means (c) Execution time of WDA-SMACOF (d) Speedup of WDA-SMACOF
Table of Contents

1 Research Backgrounds
2 Related Work
3 Research Challenges
4 Harp
   - Collective Communication Abstractions
   - Layered Architecture
5 Machine Learning Library
   - K-means Clustering & WDA-SMACOF
   - LDA
6 Conclusion
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 - $\alpha$, $\beta$, 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
            \( k' \sim p(z_{ij}|z^{-ij}, w) \propto \frac{N_{wk}^{-ij} + \beta}{\sum_w N_{wk}^{-ij} + V \beta} \left( N^{-ij}_{kj} + \alpha \right) \)
            // 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

Data Parallelism

- Global Model
  - Worker
    - Local Model
      - Training Data 1
      - Training Data 2
      - Training Data 3
      - Training Data 4

Model Parallelism

- Global Model 1
  - Worker
    - Global Model 2
    - Global Model 3
    - Global Model 4

Model Data Update between Parallel Workers or Client/Server

Global Model Rotation
Synchronized Method vs. Asynchronous Method in LDA

**Asynchronous Communication**
- Sample a word
- Send the word
- Send the word
- Send the word
- Send the word

**Synchronized Communication**
- Sample words
- Sample words
- Sample words
- Update words
- Update words
- Update words

**Threading**
- Computation
- Communication

Time
# LDA Work Using CGS Algorithm

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

Note: “D. P.” refers to Data Parallelism. “M. P.” refers to Model Parallelism.
Power-law Distribution

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

Ida-lgs
(use syncLocalWithGlobal & syncGlobalWithLocal)

Ida-rtt
(use rotateGlobal)
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
A Collective Communication Layer
Machine Learning Library
LDA

Training Datasets Used In LDA Experiments

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

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

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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Details</th>
</tr>
</thead>
</table>
| **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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Details</th>
</tr>
</thead>
</table>
| **rtt**        | - “lda-rtt” impl.  
                 - Speed comparable with Petuum on “clueweb” but 3.9 times faster on “bigram” 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) 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
A Collective Communication Layer

Machine Learning Library

LDA

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

1. Research Backgrounds
2. Related Work
3. Research Challenges
4. Harp
5. Machine Learning Library
6. Conclusion
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

References II

List of Publications


