

# Large-scale Linear Optimization through Machine Learning

JungAh Hong  
20143729

## 1. INTRODUCTION

Recent advances in cloud and distributed computing enabled Big Data processing. Today, processing big data generally falls under the category of statistical inferencing, e.g., predicting congestion with traffic data in the road network. The bottleneck is mainly the system I/O capacity for reading and writing data. The computation applied to individual items is relatively simple compared to the amount of data to process.

However, big data processing will ultimately require large scale optimizations in the future. For example, one might want to optimize the traffic light system of an entire city using predictions. Unfortunately, the traditional optimization algorithms are fundamentally hard to parallelize. These algorithms are hard to scale and cannot run on existing cloud environment.

In recent years, several abstractions using belief propagation (BP), a popular class of distributed machine learning algorithm, to parallelize linear optimizations have been proposed [1, 6]. Several well known optimization problems - the matching problem [4] and min-cost network flow problem [2] - were relaxed to a Linear Program (LP) then solved through BP in certain cases.

The goal of this research is to develop a practical system that runs on modern cloud environments, such as Amazon EC2, for parallel, distributed optimizations on maximum weight matching (MWM) problems. We plan to enhance recent theoretical developments on BP for large-scale linear optimizations and develop a cloud-based software to solve MWM problems using the power of large-scale cloud computing.

To deal with implementation issues on distributed environment, we used ZooKeeper, a centralized service for maintaining configuration information and providing distributed synchronization and group services [5].

## 2. BACKGROUND

### 2.1 Belief Propagation

Belief Propagation, also known as **sum-product message passing** is a message passing algorithm for performing inference on graphical models. Each edge asso-

ciates a message with each direction, which is updated using incoming messages. Messages are updated until convergence, and then beliefs are calculated from each node.

### 2.2 Simplified Max-product for weighted matching

- **(INIT)** Set  $t=0$  and initialize each  $a_{i \rightarrow j}^0 = 0$
- **(ITER)** Iteratively compute new messages until convergence as follows: ( $y_+ = \max(0, y)$ )
$$a_{i \rightarrow j}^{t+1} = \max_{k \in N(i)-j} (w_{ik} - a_{k \rightarrow i}^t)_+$$
- **(ESTIM)** Upon convergence, output estimate  $\hat{x}_{(i,j)}$  is, respectively,  $>$ ,  $<$ , or  $= w_{ij}$ .

### 2.3 ZooKeeper

ZooKeeper is a software project of the Apache Software Foundation, providing an open source distributed configuration service, synchronization service, and naming registry for large distributed systems.

ZooKeeper nodes store their data in a hierarchical name space, much like a file system or a tree data structure. Clients can read and write from/to the nodes and in this way have a shared configuration service.

ZooKeeper is used in many applications such as naming service, configuration management, leader election, etc.

## 3. DESIGN

### 3.1 Distributed data graph

Efficiently partitioning the data graph in the distributed environment requires balancing computation, communication, and storage. Therefore, we need to construct balanced subgraphs that minimize number of edges/nodes that cross between machines.

PowerGraph achieves balanced partitioning by considering the strongest feature of natural graphs: having highly skewed power-law degree distributions. [3] We used the same partitioning strategy and stored each part as a separate file on a distributed storage system (e.g., HDFS, Amazon S3).

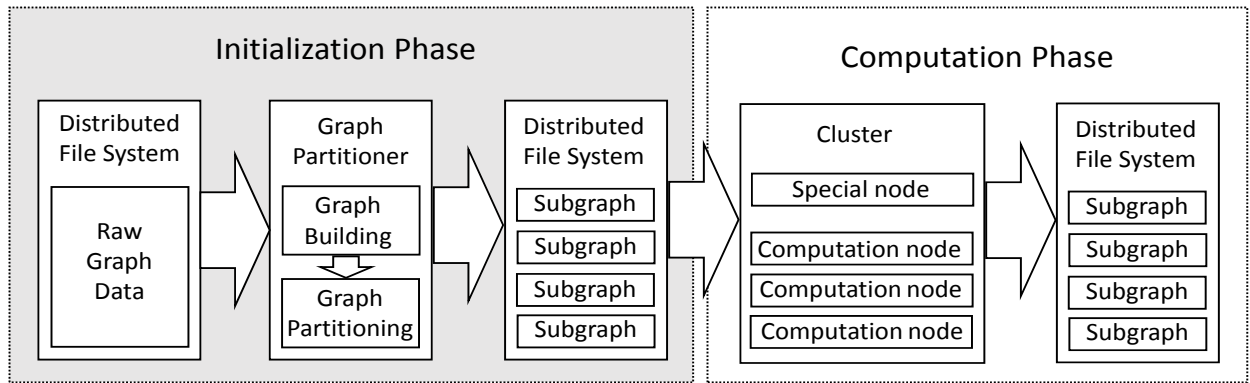


Figure 1: A high level overview of the system

### 3.2 Configuration management

The system needs management during the computation. At initialization phase, the graph is torn down to several atom graphs and mapped to each machines. Machines are assigned by ZooKeeper. A subset of nodes are assigned to particular tasks (e.g., shared data table), while the remaining nodes are assigned to do the computation. Some idle nodes remain in a pool which can be used as needed (e.g., a node might be registered as a standby for multiple active nodes).

ZooKeeper maintains overall system status and stays reliable when the fault occurs. Thus, the system would remain safe over the fault management by ZooKeeper.

### 3.3 Scheduler

The scheduler represents a dynamic list of tasks to be executed. Since new messages are calculated from the previous incoming messages, if a message is updated, the messages on the connected edges need update as well.

The schedule management is also designed with ZooKeeper. ZooKeeper maintains consistency between parallel updates with distributed locks. The computation iterates until convergence. ZooKeeper terminates the computation from all the nodes when the system reaches convergence.

### 3.4 System design

In Fig. 1, we provide a high-level overview of the system. The computation begins by constructing subgraph representation on a Distributed File System (DFS).

Each instance is executed on each machine. Some nodes are assigned to particular tasks and the rest of them are assigned to computation.

## 4. CONCLUSIONS

We identified the limitation in existing optimization algorithms that they're structurally not parallelizable. We used BP, a message passing machine learning algo-

rithm has parallel structure, to solve the MWM problem in distributed environment.

The abstraction uses a **data graph** as a computational model. Updates are done with the local computation on each vertex. Parallel **scheduler** manages the scheduling of dynamic iterative parallel computation.

To manage the task/graph assignment, scheduler and faults we used ZooKeeper, the service enables easier management of distributed configuration and synchronization.

Since ZooKeeper makes dns-based assignment, the load balancing still remains challenging.

## 5. REFERENCES

- [1] M. C. Andrew E. Gelfand, Jinwoo Shin. Belief propagation for linear programming. *ISIT*, pages 2249–2253, 2013.
- [2] Y. W. D. Gamarnik, D. Shah. "belief propagation for min-cost network flow: convergence & correctness". *SODA*, pages 279–292, 2010.
- [3] Y. L. H. G. D. B. C. G. J, Gonzalez. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX conference on Operating systems design and implementation*. OSDI, 2012.
- [4] M. S. M. bayati, D. shah. Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251, 2008.
- [5] F. P. J. B. R. Patrick Hunt, Mahadev Konar. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. USENIX, 2010.
- [6] A. W. Sujay Sanghavi, Dmitry Malioutov. Belief propagation and lp relaxation for weighted matching in general graphs. *IEEE Transactions on Information Theory*, 57(4), 2011.