Indexing data-oriented overlay networks using belief propagation *

DANNY BICKSON, DANNY DOLEV, YAIR WEISS
The Hebrew University of Jerusalem (HUJI), Israel

KARL ABERER, MANFRED HAUSWIRTH
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

In this paper we discuss the problem of data-oriented partitioning in large-scale overlay networks, as required by peer-to-peer databases or by peer-to-peer information retrieval. The goal is to partition a large set of nodes into k partitions with the additional requirement of meeting certain load-balancing constraints without global knowledge of the network's parameters, i.e., the desired number of partitions and the partition distribution function are not known in advance and change dynamically as the network evolves. This key problem in large-scale decentralized systems has so far received only very limited attention. The novel contributions described in the following are (1) the definition of a distributed algorithm for local estimation of the partitioning distribution function, which does not preclude the network's topology, and (2) a distributed method for performing the actual partitioning. As additional advantages, the algorithms do not require global knowledge and are completely decentralized, thus suitable for Peer-to-peer networks. Both algorithms are based on the max-product belief propagation algorithm and give exact results on trees, and sufficiently accurate approximations on graphs containing cycles. We show the accuracy of the proposed algorithms in terms of the number of nodes per partition and the good load balancing of partitions in the network by simulation. Our algorithms are scalable and the accuracy of the partitioning improves with larger network sizes. Having shown the efficiency of our proposed algorithms, we discuss a natural application for our algorithm in the data-oriented P2P system P-Grid (http://www.p-grid.org/). Using P-Grid's underlying tree abstraction, we can apply our algorithm recursively to achieve optimal partitioning results in short times relative to the tree diameter.

*This work was supported (in part) by the European project Evergrow No 001935.

1 Introduction

Currently the standard way of indexing in nearly all overlay networks is to compute uniformly distributed keys from the data to be indexed, e.g., by using standard hashing. With some simple additional partition assignment and maintenance algorithms (joining/leaving of nodes), this ensures that the key space is evenly distributed among the participating nodes, i.e., the storage load is balanced among the peers. This standard model for overlay networks also assumes that peers essentially join and leave in a sequential way and the resulting maintenance schemes to repair inconsistencies or re-balance load essentially correspond to updating database indexing structures which has been studied extensively in the literature.

Approaches following this basic strategy work fine for scenarios where semantic relations among the data items are irrelevant and only very simple query predicates, i.e., equality, have to be supported. A typical example for such applications would be a naming service, statically binding a set of attributes to an identifier. However, in typical data-oriented scenarios the standard strategy outlined above, will no longer work or become very inefficient. In data-oriented applications, for example, re-indexing due to a change in the data distribution, or a new data field to be indexed, or the need to preserve key-ordering relations implies that uniform hashing cannot be applied anymore as it destroys these properties. In consequence the distribution of keys will be unknown, most likely be skewed, and a more powerful, explicit partitioning scheme ensuring load-balancing will be required. Additionally, the indexing/partitioning strategy should support a high degree of parallelism to minimize latency.

Adaptive Eager Partitioning [2] (AEP) is an efficient, completely decentralized, parallel, partitioning algorithm which meets these requirements. It uses an analytically proven strategy, verified by simulation and through an implementation in the P-Grid overlay network [1] (http://www.p-grid.org/), to estimate the key distribution based on local knowledge and ensures storage and replication load balancing, i.e., each data partition contains approximately the same number of keys and approximately the same number of peers (nodes) are responsible for each partition. We have used the implementation in P-Grid as a proof-of-concept, but the essential elements of AEP are applicable to all overlay networks using fixed key space partitioning schemes, for example, CAN [9] or Pastry [10].

In this paper we present an alternative strategy based on belief propagation to achieve the same goal, i.e., to partition a large set of nodes into

k partitions. By applying this recursively we ensure routing consistency and meet AEP's storage and replication load-balancing constraints. We do not assume global knowledge of the network's parameters, i.e., the desired number of partitions and the partition distribution function are not known in advance and change dynamically as the network evolves. The motivation for this work was to have another generally applicable strategy at hand and to test and compare it with AEP.

1.1 Local estimation of the data distribution

As the first step in the approach we need to estimate the distribution of the data set to be indexed. As we assume a P2P environment, we cannot assume global knowledge of all the the data. Thus we have to estimate the distribution as good as possible based on local information. The following example illustrates this problem: Assume nodes index dictionary words, and we have 26 partitions corresponding to the letters A–Z. We would like to estimate what is the fraction of nodes needed to store words starting with each letter. This is highly important especially for skewed distributions where some partitions require more storage space relative to the others.

Given a set of $n+1$ nodes, where each node has small subset of neighbors and its local stored keys. We assume that each key belongs to exactly one partition and each node should have an estimation of the partitioning distribution function. Formally, we would like each node to calculate a vector of size k, where k is the number of different partitions, such that $p_1, \ldots, p_k$ is the percentage of keys in partition i and $\sum_{i=1}^{k} p_i = 1$.

1.2 K-Partitioning Problem

Given a set P of $n+1$ peers which hold keys from a keyspace K, we would like to partition the set into $(1, \ldots, k)$ sets such that the load measured in number of data keys related to the partitions $1, \ldots, k$ is expressed in a probability distribution $p_1, \ldots, p_k$, which is termed *partitioning function*.

The partitions we would like to achieve should have the following properties:

1. Proportional replication: Each peer has to decide for one partition such that (in expectation) a fraction p_i of the peers decides for partition i.
2. Referential integrity:

1. The graph has to be connected, i.e., each peer has sufficient links to reach any other partition (in-)directly.
2. Each peer strives to maximize the number of neighbors from different partitions under its degree constraint.

Note that without the proportional replication requirement, the peers could decide for a partition based on the partitions probability function, and in expectation achieve the desired partitioning. However, this would not ensure approximately uniform availability of all the data. The referential integrity function ensures routing consistency and can be exploited to achieve better load balancing of partitions.

For being able to evaluate the quality of a solution for the above problems we need to define some metrics. As for the estimation problem we might take the mean square error of an estimate relative to the correct partitioning distribution function.

In the K-partitioning problem we have two metrics that might be contradicting. First, we want to have an exact numeric partitioning of nodes as close as possible to the partitioning distribution function. Here we can take the mean square error as well. Another metric is the link satisfaction ratio, where we would like to minimize the number of neighboring nodes which are partitioned into a common partition. The first metric arises from the need to allocate a proportional number of nodes based on the storage needs. The second metric is important in terms of load balancing where we would like to prevent unbalanced placement of a partition in a small neighborhood of nodes. This is bad in terms of resiliency to failures, and network load.

2 Proposed algorithms

For solving the two problems of distributed partitioning function estimation and the extended partitioning problem we will use the max-product belief propagation algorithm which we will briefly outline in this section. Then we present the technical details needed for constructing both algorithms. First, we run the distributed estimation algorithm to equip nodes with local knowledge of the partitioning distribution function and then we run the K-partitioning algorithm using the result of the estimation algorithm as its input.

2.1 Graphical models for data indexing

An undirected graphical model G consists of a set of vertices V and a set of edges E connecting them. Each vertex v_i is associated with a random variable x_i. We assume that the joint probability distribution factors into a product of terms involving node pairs and single nodes. These factors are called edge potentials $\psi_{ij}(x_i, x_j)$ and local (or self) potentials $\psi_{ii}(x_i)$.

The max-product belief propagation algorithm [7] is a distributed inference algorithm that enables us to calculate the marginal probabilities of the nodes, otherwise known as the "beliefs." It is a distributed message-passing algorithm and is therefore suitable for communication networks [4, 5, 8]. The Belief Propagation(BP) algorithm gives exact results on trees. We have no guarantee for the algorithm performance on graphs with cycles. As P-Grid is a tree and has no cycles this poses no problems. The input to the BP algorithm is a graphical model with self potentials $\psi_{ii}(x_i)$ and edge potentials $\psi_{ij}(x_i, x_j)$. The output of the algorithm is the vector of node beliefs (posteriori probabilities). The algorithm is an iterative distributed message passing algorithm where messages sent between nodes are determined by the following update rule:

$$m_{ij}(x_j)^{(t+1)} = \alpha \max_{x_i} \psi_{ij}(x_i, x_j)\psi_{ii}(x_i) \prod_{x_k \in N(x_i) - x_j} m_{ki}^{(t)}(x_i) \tag{1}$$

where $m_{ij}(x_j)$ is a message sent from node x_i to node x_j, α is a normalization factor and $N(x_i)$ is the set of neighbors of node x_i. We initialize the messages at the first round uniformly. Finally each node calculates the belief:

$$bel(x_i) = \alpha\psi_{ii}(x_i) \prod_{x_j \in N(x_i)} m_{ji}(x_i) \tag{2}$$

The algorithm converges when the node receives identical messages from all neighbors for two consecutive rounds. In networks containing cycles the algorithm might not converge. However, in practice, there are several applications where the algorithm produces very good results even for graphs with cycles, for example, in the case of Turbo codes.

It is known that if we use the max-product algorithm we can find an optimal X^* that maximizes the probability $P(x)$, if the topology has no cycles, and we can find a strong local maximum in case the graph has cycles [11]. In other words, our algorithm solves the problem optimally on trees and gives a good approximation of a graph containing cycles.

Based on equation 1, we calculate the self potentials $\psi_{ii}(x_i)$ and edge potentials $\psi_{ij}(x_i, x_j)$ for a given cost function.

2.2 Algorithm for estimation of the partition distribution function

First we outline a simpler version of the algorithm for tree topologies. In round $(t+1)$ each node i sends a message $m_{i,j}(x_j)$ to each of its neighbors j by the following rules:

$$m_{ij}^{(t+1)}(x_j) = \sum_{k \in N(x_i) - x_j} m_{ki}^{(t)}(x_i) \tag{3}$$

where the initialization of $m_{ij}^{(0)(x_j)}$ is done by assigning the number of keys node i holds in each partition. For example, if node i has 3 keys which are related to partition 0, the message will contain the value 3 in position 0 of the vector. Finally, we estimate the partitioning distribution function:

$$bel(x_i) = \sum_{k \in N(x_i)} m_{ki}^{(t)}(x_i) \tag{4}$$

Basically we have here an aggregation procedure along the tree, where each node aggregates the partition estimation from all of its neighbors on the subtree rooted in the node, and forwards it upstream in the tree. After *diam* rounds, where *diam* is the tree diameter, the algorithm converges and gives an exact result.

Since overlay network topologies may contain cycles, we can apply the consensus propagation algorithm [6] recently proposed by Moallemi and van Roy to avoid over-counting of the partitions because of cycles. Thus our approach can be generalized to any overlay network topology. Due to space constraints we omit a description of this algorithm and refer reader to [6].

2.3 The K-partitioning algorithm

As already mentioned, we use the max-product BP algorithm for solving the K-partitioning problem. In order to use the BP algorithm, we need to initialize the local potentials and edge potentials of the nodes.

For the self potentials, each node draws locally a random partition i out of the partitioning distribution function $(p_1, p_2, \ldots, p_k)$ and initializes the local potentials vector to have 1 in cell i and ε elsewhere.

For the edge potentials, we use the following matrix:[1]

$$\forall e_{i,j} \in E \ \psi_{ij}(x_i, x_j) = \begin{bmatrix} \varepsilon & k' & k' & \dots & k' \\ k' & \varepsilon & k' & \dots & k' \\ k' & k' & \varepsilon & \dots & k' \\ \dots & \dots & \dots & \dots & \dots \\ k' & k' & k' & \dots & \varepsilon \end{bmatrix} \tag{5}$$

Each matrix $\psi_{ij}(x_i, x_j)$ corresponds to the edge e_{ij}. The K rows indicate the probabilities to choose a partition for node i and the K columns indicate the probabilities to choose partitions of node j. Thus the matrix $\psi_{ij}(x_i, x_j)$ is the Cartesian product of both nodes possible partitions. We build the edge potentials by the pairwise constraint, that two neighboring nodes should not have the same partition. For that, we assign a negligible probability (ε) to the main diagonal. The other states have a uniform probability.

A related work of a load-balanced graph coloring using the BP algorithm is done by Saad et. al [3]. The major differences are in the construction used. In a nutshell, Saad uses centralized computation on a factor graph while we use distributed computation over the communication network. Saad's coloring is balanced while we have desired fraction of nodes from each color. Finally, we use the local potentials to reflect the partitioning function while previous works on this area initialize the local potentials to have a uniform probability.

2.4 The full algorithm

Input: A set of nodes where each node has knowledge of a small subset of other nodes. Each node knows only his locally stored keys as well as the partition to which they belong.
Output: A partitioning of the nodes into K partitions where the number of nodes in each partition is based on the partitioning distribution function. Each node has an estimation of the partitioning distribution function.

Each node follows the following steps:

1. Estimate the partition distribution function using the consensus propagation algorithm.
    1. Input: Each node has knowledge of a subset of other nodes, and the keys it currently stores.

[1] Each row is normalized to get a sum of one. $k' = 1/(k-1)$

2. Initialize $m_{ij}^{(0)}(x_j)$ to the key count from each partition.
3. Run t rounds of the aggregation algorithm (equation 3) .
4. Estimate the partition distribution function (equation 4).
5. Output: local estimation of the partitioning distribution function.

2. Run the K-partitioning algorithm.
    1. Input: estimation of the partitioning distribution function.
    2. Initialize $m_{ij}^{(0)}(x_j)$ by randomly drawing a partition from the partitioning distribution function.
    3. Run the max product belief propagation for t rounds (equations 1 and 5).
    4. Calculate the belief (equation 2).
    5. Select the maximal entry in the belief vector (the decided partition number).

Properties of the algorithm. Regarding running time, when running on tree topologies, the algorithm converges and gives an exact result in time $diam$ where $diam$ is the tree diameter. On topologies containing cycles the algorithm does not always converge, thus we have to stop it after t rounds. As a heuristic we take $t = log(n)$ where n is the number of overlay nodes. Weiss in [11] shows that on a graph with cycles we get a strong local maximum in case the algorithm converges, in other words a good approximation to the optimal.

When the algorithm does not converge, we get an approximation to the optimal solution. In this case we are only able to show using simulations that the algorithm gives a good approximation to the optimal.

As for the number of messages, in each round each node sends messages to all of its neighbors. Thus we have a total of tdn messages where d is the average node degree. All messages are of size k where k is the number of partitions.

3 Experimental results

For testing our algorithm, we used two typical topologies: a random graph and a GT-ITM transit stub topology. Both topologies have cycles. In the following we show the results on the random graph only, since both topologies rendered similar results.

Figures 1(a)–1(d) show results of algorithm runs for several partitioning functions. We would like to point out that our algorithm supports any desired partitioning distribution function.

For the partitioning estimation function, we assumed an underlying tree topology, and thus received an exact estimation of the partitioning distribution function. This assumption can be justified since we used the P-Grid system as a natural application that might use the partitioning algorithms described in this paper. In the P-Grid system we have an underlying tree structure that can be used for the estimation procedure. An area of future research might be to examine the algorithm's correctness for various other topologies. For the rest of this section we analyze the K-partitioning algorithm results.

The BP algorithm generally did not converge, thus we chose to limit the number of algorithm rounds to d (the node degree). Limiting the number of rounds is a heuristic which is discussed in the previous section. In each round, we send d messages from each node to its neighbors. Thus in total we have nd^2 messages. Assuming that $d = k = \log(n)$, we have a total number of $n\log(n)^2$ messages.

In the experiments we used networks of up to 500 nodes, in which each node had 10-12 random neighbors (approximately $\log(n)$). Each node had to choose from among 10-12 partitions. Under all policies, each node tries to have as many neighbors in as many partitions as possible. To measure the number of neighbors which break the referential integrity condition we define the *unsatisfied link ratio* which is the ratio of graph edges connecting two nodes which have the same partition to the total number of edges.

We have also used several partitioning distribution functions in the experiments: Balanced partitioning means that all partitions should have the same number of nodes; exponentially skewed partitions use an exponential distribution; and linear partitioning means that the size of each partition grows linearly relative to the previous partition.

4 Conclusions and future work

In this paper we proposed an algorithm for solving the extended partitioning problem in overlay networks. As shown by simulation our algorithm performs well and scales to large networks. An area for future work might be to optimize the transfer of node keys to the nearest matching partition.

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-organizing structured P2P system. *SIGMOD Record*, 32(3), 2003.

[2] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing Data-oriented Overlay Networks. In *VLDB*, 2005.

[3] S. Bounkong, J. van Mourik, and D. Saad. Coloring Random Graphs and Maximizing Local Diversity. In *cond-mat/0507579*, 2005.

[4] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication networks. In *In Proceedings of the 19th Conference on Uncertainty in AI*, 2003.

[5] Ihler, Fisher, Moses, and Willsky. Nonparametric belief propagation for self-calibration in sensor networks. In *Information Processing in Sensor Networks*, 2004.

[6] C. C. Moallemi and B. V. Roy. Consensus propoagation. In *NIPS 2005*, 2005.

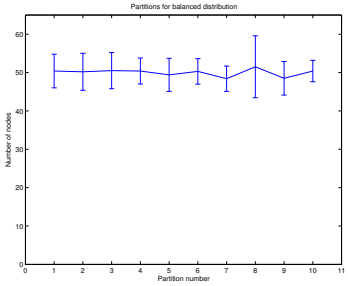
[7] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of Uncertainty in AI*, 1999.

[8] M. Paskin and C. Guestrin. Robust probabilistic inference in distributed systems. In *In the Twentieth Conference on Uncertainty in Art ficial Intelligence, Banff, Canada*, 2004.

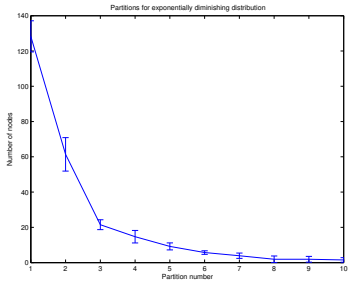
[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, 2001.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

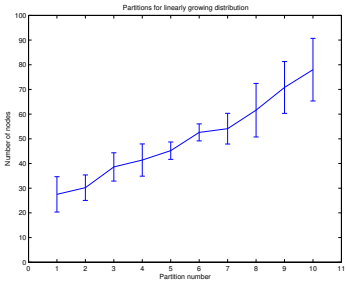
[11] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2), 2001.



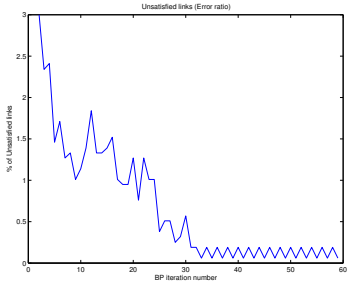
(a) Balanced partitioning into 10 partitions on a network of 500 nodes. Each partition should have 50 nodes in the optimal solution. This figure results for an average of 10 runs on 10 random graphs is shown. The average smallest partition had 48 nodes in the worst case and the largest partition had 52 nodes. The error bars show the variance of the partition size distribution. The average unsatisfied link ratio is less than 0.2%



(b) Skewed distribution partitioning into 10 partitions on a network of 500 nodes. This figure shows the results for an average of 10 runs on 10 different random graphs is shown. The input partitioning distribution function is $(1/2, 1/4, 1/8, \ldots, 1/1024)$.



(c) Linearly increasing distribution of 10 partitions for a network of 500 nodes. This figure shows the results of an average run on 10 random graphs.



(d) The quality of a balanced partitioning where n=500, k=10, d=10. An unsatisfied link means that two nodes sharing the same edge decided on the same partition. Note that the BP algorithm fluctuates toward the optimal solution.