Classifying Documents by Distributed P2P Clustering

Martin Eisenhardt Wolfgang Müller Andreas Henrich

Chair of Applied Computer Science I
University of Bayreuth, Germany
{eisenhardt|mueller2|henrich}@uni-bayreuth.de

Abstract: Clustering documents into classes is an important task in many Information Retrieval (IR) systems. This achieved grouping enables a description of the contents of the document collection in terms of the classes the documents fall into. The compactness of such a description is even more desirable in cases where the document collection is spread across different computers and locations; document classes can then be used to describe each partial document collection in a conveniently short form that can easily be exchanged with other nodes on the network. Unfortunately, most clustering schemes cannot easily be distributed. Additionally, the costs of transferring all data to a central clustering service are prohibitive in large-scale systems. In this paper, we introduce an approach which is capable of classifying documents that are distributed across a Peer-to-Peer (P2P) network. We present measurements taken on a P2P network using synthetic and real-world data sets.

1 Motivation

In P2P IR systems and Distributed IR Systems, an important problem is the routing of queries or the selection of sources which might contain relevant documents. Some approaches in this respect are based on compact representations of adjacent peers' document collections or the documents maintained on the reachable source nodes. In this context, an interesting approach is to determine a consistent clustering of all documents in the network, and to acquire knowledge about which node contains how many documents falling into each cluster. Then, for a given query, the querying node can identify the most promising clusters by that compact representation of other peers' collections; the query can then be routed precisely to nodes potentially containing relevant documents.

In the scenario above, an efficient algorithm is needed to determine the clustering of the global document collection. The present paper will describe and assess such an algorithm. The concrete motivation behind this approach stems from our research on large-scale P2P IR networks (cf. [EH02]). In environments like this, the costs of transferring documents and/or data to a central clustering service become prohibitive; in some cases, it might even be infeasible to transfer the data, as Skillicorn points out (see [Ski01]). Therefore, a distributed clustering algorithm is needed.

2 Background

Our approach is based on two well-known algorithms from the areas of statistics and distributed systems. We use the *k*-means clustering algorithm to do the actual categorization of the documents. A probe/echo mechanism is used to distribute the task through the P2P network and propagate the results back to the initiator of the clustering.

The *k*-means Clustering Algorithm The *k*-means clustering algorithm is a non-hierarchical approach to clustering (see the box titled *Algorithm 1*). Its input parameters are a number κ of desired clusters, a set D of n-dimensional data objects d_i, and a initial set C of n-dimensional cluster centroids c_j where $d_i, c_j \in \mathbb{R}^n$; $c_{i,l}$ and $d_{j,l}$ depict the n single vector elements. The clustering algorithm follows a very straight-forward approach: in each iteration, the distances of each $d_i \in D$ to all $c_j \in C$ are computed. Each d_i is allocated to the nearest c_j. Then, every centroid c_j is recalculated as the centroid of all data objects d_i allocated to it.

There are a few mostly equivalent termination conditions for the *k*-means algorithm. One of the most widely used is that the algorithm terminates if the sum of all mean square errors (MSE) within each cluster does no longer decrease from iteration to iteration.

Note that the function $dist(d_i, c_j)$ can be chosen according to the data set you are clustering: a common class of distance metrics are the Minkowski metrics $dist_{mink}(c_i, d_j) = \sqrt[m]{\sum\limits_l |d_{i,l} - c_{j,l}|^m}$ where the parameter $m \in \mathbb{N}$ can be set to generate a specific metric, e.g. the Manhattan metric ($m = 1$) or the Euclidean metric ($m = 2$); the latter metric was used in our experiments. Other metrics (f.e. cosine metric) are also possible. Various methods exist for the generation of initial cluster centroids, ranging from quite obvious ones (just take $|C|$ randomly chosen documents as the initial centroids) to sophisticated ones that lead to a faster convergation and termination of the algorithm (see [BF98]).

As Modha and Dhillon point out in [DM00], the *k*-means algorithm is inherently data parallel. Since it involves no hierarchical scheme, the only data that must be shared among the nodes are the cluster centroids; all locally refined centroids can then be merged at the node which initiated the clustering; this node decides according to a termination condition whether another iteration of the *k*-means algorithm is necessary. The advantage of this approach is that the transfer of relatively few centroids is feasible as opposed to transferring whole data sets. Additionally, the *k*-means algorithm is well-suited for the clustering of documents, as is shown in [SC02].

The Probe/Echo mechanism The Probe/Echo mechanism was first described by E. J. H. Chang in [Cha82]. It is an algorithm to distribute a piece of information across a general graph, in our case a computer network. We want every node to receive the information only once to reduce the amount of messages sent and the number of processing steps. The algorithm implicitly constructs a spanning tree of the graph. The message complexity on a graph $G = (V, E)$ with $|E|$ edges is $2|E|$.

The initiator of the probe/echo mechanism marks itself as both `engaged` and `initiator` and sends a `PROBE` message to all its neighbours. When a node receives the first `PROBE`,

Algorithm 1: The k-means clustering algorithm.

```
Input   : $D \subset \mathbb{R}^n$ – the set of data points $d_i \in D$.
Input   : $\kappa$ – the number of clusters the algorithm should calculate.
Output  : $C \subset \mathbb{R}^n$ – the set of cluster centroids $c_j \in C$.
Data    : $M \subset 2^D$ – the set of sets $m_j$ where each $m_j$ contains the $d_i$ closest to $c_j$.

begin
    $MSE \leftarrow 0, MSE_{old} \leftarrow \infty$;
    repeat
        foreach $m_j \in M$ do $m_j \leftarrow \emptyset$;        // initialize all clusters as empty sets
        foreach $d_i \in D$ do
            Find closest $c_j \in C$;
            $m_j \leftarrow m_j \cup \{d_i\}$;        // assign each object to its nearest cluster
        foreach $c_j \in C$ do $c_j \leftarrow \frac{1}{|m_j|} \sum_{d_i \in m_j} d_i$;        // recalculate cluster centroids
        $MSE_{old} \leftarrow MSE$;
        $MSE \leftarrow \sum_j \frac{1}{|c_j|} \sum_{d_i \in m_j} (dist(d_i, c_j))^2$;        // calculate cluster quality
    until $|MSE_{old} - MSE| < \varepsilon$
end
```

it marks itself as engaged and sends PROBEs to all adjacent nodes. All subsequent received PROBEs do not cause the node to send an additional PROBE to its neighbours. After a node has received PROBE or ECHO messages from all its neighbours it marks itself as not engaged and sends an ECHO to the node it received its first PROBE from. When the initiator has received ECHOs from all its neighbours, the algorithm terminates.

3 A Distributed Algorithm to Cluster Documents

Based on the two algorithms above, we have developed and implemented an algorithm for the distributed clustering of documents (see box *Algorithm 2*). The initiator of the clustering guesses an initial set of cluster centroids and sends these centroids along with an PROBE to all its neighbouring peers. Every peer receiving a PROBE proceeds with resending the PROBE to all its neighbours except the one it received the PROBE from (the *pred*ecessor). Then, a k-means clustering on the locally available documents is done. On receiving an ECHO from an adjacent peer, the peer merges the clustering done by the remote peer C_p with the clustering resultant from the local clustering C_l; this is done by calculating a weighted mean taking into account how many documents each peer grouped into the respective clusters (W_l and W_p map clusters to their associated weights). After having received either a PROBE or an ECHO from every adjacent peer, the peer sends an ECHO containing the merged cluster centroids and the weighting to the peer it received the first PROBE from. When the initiator has received ECHOs from all its adjacent peers, it has

Algorithm 2: Our distributed clustering algorithm.

```
Input : Neighbours – the set of all neighbouring nodes.
Input : $D \subset \mathbb{R}^n$ – the data points $d_i \in D$ on the local peer.
Data  : $(C_p, W_p)$ – remote set of cluster centroids and associated weights.
Data  : $(C_l, W_l)$ – local set of cluster centroids and associated weights.
begin
    receive (PROBE, $C_p$) or (ECHO, $C_p$, $W_p$) from $p \in$ Neighbours;
    if PROBE then
        if $\neg engaged$ then
            $engaged \leftarrow$ true; $received \leftarrow 1$; $pred \leftarrow p$;   // received first PROBE
            send (PROBE, $C_p$) to Neighbours$\setminus\{pred\}$;
            $(C_l, W_l) \leftarrow kmeans(C_p, D)$;   // one local iteration of k-means
        else $received \leftarrow received + 1$;   // received additional PROBE
    if ECHO then
        $received \leftarrow received + 1$;   // received ECHO
        $(C_l, W_l) \leftarrow mergeResults(C_l, W_l, C_p, W_p)$;   // merge local and received results
    if $received = |Neighbours|$ then
        $engaged \leftarrow$ false;   // termination condition fulfilled
        if $initiator$ then terminate;
        else send (ECHO, $C_l$, $W_l$) to pred;
end
```

complete information on the current iteration of the k-means algorithm. Therefore, this iteration is complete and terminates. Full k-means clustering typically requires more than one iteration. Thus, based on a condition as described in section 2, the initiator decides whether the currently achieved clustering needs to be improved by a further iteration. In those subsequent iterations, the initiator sends the resultant cluster centroids C_l of the last iteration as the new “guess” to its neighbours.

4 Experimental Setup

Hardware For our experiments, we used the computers in our laboratory and two other computer laboratories on campus. The hardware used is quite heterogeneous, ranging from Intel Pentium 4 processors at various speeds (≈ 1.4GHz – 2.0 GHz) to Athlon MPs at 1.5 GHz. The nodes were equipped with different amounts of main memory (≈ 256 MBytes – 1024 MBytes). The three sites (our laboratory and the two laboratories on campus) were interconnected by the university's Fast Ethernet backbone; each site had a dedicated Fast Ethernet switch to route local messages and to connect the site to the backbone.

Data Sets We used synthetic data as well as real-world data. The synthetic data sets consist of evenly distributed random numbers $\in [0; 1000]$. We used data sets with

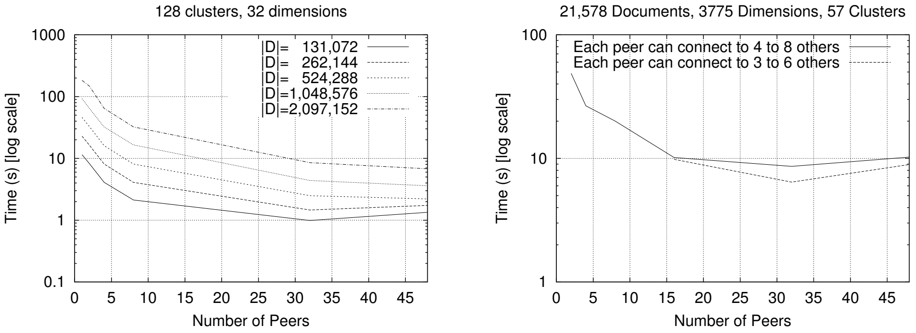

Figure 1: The left figure shows the times for one iteration of our Algorithm 2, measured on data sets with $2^{17} - 2^{21}$ (131,072 - 2,097,152) tupels with 32 dimensions. The right figure shows the times for clustering the Reuters-21578 collection (21,578 documents with 3,775 features).

$2^{17} \leq |D| \leq 2^{21}$ data points and $2 \leq n \leq 32$ dimensions or features. Our real-world data set is an index of the Reuters-21578 news article collection. It contains 21,578 documents; $n = 3,775$ features remained after stopword elimination, elimination of extremely rare terms, and Porter stemming.

Measurements Each measurement was performed on 2, 4, 8, 16, 32 and 48 peers. The measurements on the synthetic data sets were additionally made with only one peer running, to achieve a baseline against which the distributed case can be compared. The synthetic data sets were clustered into $\kappa = 8, 32, 128$ clusters, whereas the real-world data set from the Reuters-21578 news article collection was clustered into 57 clusters, which seems to be a reasonable value, since the collection has been classified by human experts into 57 clusters with at least 20 documents each[1]. For our experiments, we allowed each peer to communicate with 4 to 8 other randomly chosen peers. We took care to make inter-sites connections as probable as intra-site connections.

5 Experimental Results

From the measurements as described above, we depicted two especially interesting results. As can be seen on the left side of figure 1, the time for clustering the synthetic data sets diminishes with an increasing number of peers. Best results can be obtained if the size of the local document collection on each peer as well as the number of desired clusters does not fall beyond a certain limit. This explains why the times for the two smaller data sets with 131,072 and 262,144 tupels increase when clustered on 48 peers instead of 32 peers. A similar observation has been made in [DM00].

Figure 1, *right*, contains the results obtained from measurements on the Reuters-21578 text collection. Clearly, our approach is capable to perform the clustering of such a collection.

[1] See `http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt` for more information.

Again, the times for one iteration are higher in the case of 48 clustering peers than they are in the case of 32 peers; in the first case, there are only $\frac{21578}{48} \approx 450$ documents per peer; with a document collection this small our algorithm shows only a limited amount of its potential. An interesting fact is that the network topology has an important impact on the times measured. When changing the connectivity—each peer now only connects to 3 – 6 other peers—the measured times for one iteration decrease, especially when 16, 32, or 48 peers participate in the distributed clusterings shown in the graph.

6 Conclusion

In this paper, we presented an algorithm for the distributed clustering of documents. We have shown that our algorithm is capable of handling real-world-sized problems and that distribution creates—even in our P2P setting—speed ups in comparison to centralized processing. For data sets such as text (many features, many documents with respect to the number of desired clusters), the savings in transfer time alone justify our distributed approach.

We find these results very encouraging for our setting of P2P IR, as in addition to the documented speedup, the processor load in each peer is dramatically reduced with respect to a client/server scenario in which clients send data to a dedicated clustering service. It can thus be safely assumed that a distributed calculation of k-means clusters can be performed as a background task in a P2P IR network. As a consequence, the achieved clustering can be used for a compact representation of peer contents in the form of histograms denoting the number of documents per peer falling into each cluster. These representations can be distributed in the network and used as a basis for source selection and query routing in a P2P network.

References

[BF98] Paul S. Bradley and Usama M. Fayyad. Refining initial points for K-Means clustering. In *Proc. 15th International Conf. on Machine Learning*, pages 91–99. Morgan Kaufmann, San Francisco, CA, 1998.

[Cha82] E. J. H. Chang. Echo Algorithms: Depth Parallel Operations on General Graphs. *IEEE Transactions on Software Engineering*, 8(4):391–401, 1982.

[DM00] Inderjit S. Dhillon and Dharmendra S. Modha. A Data-Clustering Algorithm on Distributed Memory Multiprocessors. In *Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD, Aug 15, 1999, San Diego, USA*, volume 1759 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2000.

[EH02] Martin Eisenhardt and Andreas Henrich. Problems and Solutions for P2P IR Systems (in german). In Sigrid Schubert, Bernd Reusch, and Norbert Jesse, editors, *Informatik 2002*, volume 19 of *LNI*, Dortmund, 2002. GI.

[SC02] Mark Sinka and David Corne. A Large Benchmark Dataset for Web Document Clustering. In *Soft Computing Systems: Design, Management and Applications, Vol. 87 of Frontiers in Artificial Intelligence and Applications*, pages 881–890, 2002.

[Ski01] David B. Skillicorn. The Case for Datacentric Grids. External technical report, Dept. of Computing and Information Science, Queen's University, Kingston, Canada, Nov. 2001.