Collaborative Hierarchical Clustering in the Browser for Scatter/Gather on the Web

Weimao Ke and Xuemei Gong
Laboratory for Information, Network & Computing Studies
College of Information Science and Technology
Drexel University, 3141 Chestnut St, Philadelphia, PA 19104
wk@drexel.edu, xg45@drexel.edu

ABSTRACT
Scatter/Gather is a powerful browsing model for exploratory information seeking. However, its potential on the web scale has not been demonstrated due to scalability challenges of interactive clustering. We have developed in previous research a two-stage method to support on-the-fly Scatter/Gather, in which an offline module pre-computes a hierarchical structure to support constant time on-line interaction. In this work, we focus on the offline hierarchy construction and develop a novel distributed approach to hierarchical agglomerative clustering (HAC). Relying on Javascript that is commonly supported by browsers, the distributed clustering method has the potential to scale with growing traffics of a site. We show in experiments that a moderate increase in the number of parallel processes (in visitors’ browsers) leads to a dramatic decrease of clustering time. This demonstrates great potentials in supporting large-scale Scatter/Gather interactions on the web. We present preliminary analysis of clustering effectiveness and a related Scatter/Gather prototype for web search.

Keywords
text clustering, Scatter/Gather, distributed computing, parallel clustering, browser server, Javascript, interactive information retrieval, exploratory search

INTRODUCTION
Information retrieval (IR) systems such as web search engines play important roles in connecting people with information. While searching is a widely accepted approach to finding information, browsing represents another basic IR paradigm. Among classic browsing models, Scatter/Gather is a unique approach in which searches can be conducted without explicit query specification (Cutting et al., 1992). Based on iterative user selection and interactive text clustering, Scatter/Gather offers a powerful tool for navigating a large, complex information space. It enables the user to explore inherent associations among documents and topics in the data, supporting learning and investigation (Hearst and Pedersen, 1996).

However, major challenges associated with clustering efficiency and scalability have hindered the adoption of Scatter/Gather in IR practice. In particular, many clustering algorithms are computationally complex. Even efficient classic methods such as k-means are of linear time complexity, far from efficient to support on-the-fly clustering on a large number of documents. The use of Scatter/Gather for web browsing is desirable but practically challenging because of the web’s scale and dynamics. Until we can properly address these challenges, real-world applications of Scatter/Gather are unlikely to emerge.

Notwithstanding its great potential in interactive IR, Scatter/Gather research has so far focused on rather small data collections. Its efficiency and effectiveness on the web scale remain unaddressed. The research aims to study scalable approaches to interactive clustering. A major objective is to identify a scalable clustering architecture that can support Scatter/Gather interactions on the evolving web. Ultimately this will lead to new development of web browsing techniques.

RELATED WORK
Scatter/Gather is a highly interactive model for collection browsing and information retrieval based on text clustering (Cutting et al., 1992). It supports progressive query specification through user-system interaction and clustering. In each Scatter/Gather iteration, the system presents to the user a set of clusters (topical groups of documents) in the information collection. The user then picks one or more clusters s/he is interested in, on which the system performs clustering again to identify new topical groups.

After each step, the information need is better clarified while the user delves into more focused subsets. The system and the user can thus achieve a better mutual understanding about what is needed and together
identify relevant information. This also helps the user explore the inherent associations among documents and topics in the information space, supporting exploratory learning (Hearst and Pedersen, 1996; Ke et al., 2009).

However, major challenges associated with clustering efficiency and scalability have hindered the adoption of Scatter/Gather in IR practice. Since the Scatter/Gather method requires on-the-fly clustering on a large data corpus, fast clustering algorithms are essential. Clustering efficiency is often more important than accuracy because it is the real-time interaction with the user that potentiates the value of Scatter/Gather (Hearst et al., 1995).

Two linear time clustering algorithms, namely the Buckshot and the Fractionation, were implemented for the original Scatter/Gather method (Cutting et al., 1992). Both algorithms have $O(kn)$ time complexity, where $k$ is the number of desired clusters and $n$ the total number of documents. As compared to the Buckshot, the Fractionation algorithm is a little slower but with higher accuracy.

Although better than a quadratic time complexity, $O(kn)$ is not fast enough for large document collections. A parallel version of the Buckshot algorithm, which achieved a $O(n \log n)$ time complexity, was further proposed and evaluated (Jensen et al., 2002).

Notwithstanding its great potential in interactive IR, Scatter/Gather research has so far focused on rather small data collections. Its efficiency and effectiveness on the web scale remain unaddressed. Our research has studied various Scatter/Gather applications and focused on supporting scalable, on-the-fly clustering (Ke et al., 2008, 2009).

**Two-stage LAIR2 Method**

In previous research, we developed a two-stage method called LAIR2 to boost on-line clustering efficiency. With a hierarchical structure pre-computed in the off-line stage, LAIR2 performs cut-tree operations in the on-line Scatter/Gather stage to identify related clusters and has constant time complexity.

LAIR2 is simple and flexible – various hierarchical or partitioning clustering methods can be employed in the off-line phase for cluster pre-computation. With constant time complexity in the on-line phase, it meets the desired efficiency for interactive Scatter/Gather clustering.

In the first phase of the LAIR2 clustering algorithm, an arbitrary agglomerative (or divisive) hierarchical algorithm can be used to construct a dendrogram (tree) of clusters (e.g., Figure 1), represented by a sequence of agglomerated pairs of data points.

In the on-line Scatter/Gather clustering phase, LAIR2 produces clusters by taking advantage of the cluster hierarchy or dendrogram constructed in the first phase. Suppose the desired number of clusters is $k$ and the number of clusters selected is $k'$, where $k' < k$. The system splits the current $k'$ selected clusters based on the precomputed dendrogram in a top-down manner. Starting from the $k'$ branches, the process moves downward to sub-branches. When a cluster pair at a certain height level is reached, we split it by removing the entry and adding its two sub-clusters. This process is repeated until $k - k'$ clusters have been split, or $k$ centroids have been identified.

This is essentially a cut-tree operation on selected subsets. Figure 1 illustrates how different numbers of clusters can be generated by cutting a tree at different heights. If, for example, the two starred (with *) branches/clusters are selected for Scatter/Gather, cutting at the dashed line in Figure 1 generates six clusters ($c_1$–$c_6$) from the dendrogram. Details about the LAIR2 algorithm can be found in Ke et al. (2008).

![Figure 1: A dendrogram illustration of LAIR2. Cut-tree operations for Scatter/Gather.](image)

**LAIR2 Efficiency in the On-line Phase**

Our previous research demonstrated the efficiency of LAIR2’s on-line clustering (Ke et al., 2008). On a data collection containing tens of thousands documents, the on-line clustering module responded within milliseconds and was several hundred times faster than a parallel Buckshot algorithm (Cutting et al., 1992; Jensen et al., 2002). When the size of data collection increased to hundreds of thousands, the clustering time of our algorithm remained at a constant level. Please refer to Ke et al. (2008) for detailed results on the efficiency of the LAIR2 algorithm and how it supported a real-time system for on-line Scatter/Gather browsing.

**LAIR2 Hierarchy Construction Challenge**

Although LAIR2 is efficient in the on-line phase, its offline component suffers from scalability challenges. A standalone LAIR2 system cannot sustain growing data volumes on the web. For the off-line stage, existing hierarchical clustering algorithms are too inefficient to deal with large-scale web data. And given constant changes, it is very challenging, if not impossible, even for an off-line method to process updates in a timely manner. Although text clustering can be parallelized on server clusters to improve efficiency, few have sufficient resources (servers) for this data-intensive processing. Here we present a novel distributed approach to tackle this challenge.
DISTRIBUTED HAC IN BROWSER

For the data-intensive off-line hierarchy construction in LAIR2, we propose a distributed (parallel) hierarchical agglomerative clustering (HAC) method. For practical applications in which server resources are limited and/or too expensive to support large-scale parallel computing, we further develop a parallel clustering framework based on Javascript, which is commonly supported by web clients (browsers). By utilizing computational powers in browsers, the clustering capacity of a web site becomes proportional to its traffic. Therefore, the distributed clustering architecture has the potential to sustain growing popularity of a site.

Distributed Hierarchical Clustering

Here we describe the proposed distributed hierarchical agglomerative clustering (dHAC) method. Text clustering usually requires a document collection be tokenized and vectorized. In this work, we focus on clustering of document vectors and assume they have been preprocessed.

The entire distributed HAC process consists of two phases. In the first phase, given available processes, a dHAC master (server) divides a collection of documents into segments and distributes each data subset to a process. Each process then independently performs HAC clustering to build a hierarchy (a binary tree) on the subset (Jain et al., 1999). The tree keeps additional information such as the centroid data point at each merge.

A process sends the hierarchical structure back to the master when an individual HAC job is done. The time complexity of HAC is \( O(n^2) \). In the first phase of dHAC, each parallel process has \( n/p \) documents. Hence, only \( O(n^2/p^2) \) time is needed in the distributed process.

After all processes finish jobs and send results back, the master server initiates the second phase to merge the subset hierarchies. It (sequentially) identifies \( p/2 \) (or \( p/2 + 1 \) if \( p \) is an odd number) closest hierarchy pairs based on distances of their root centroid vectors and distributes the pairs to \( p/2 \) (or \( p/2 + 1 \)) processes for parallel merging of the subsets.

Subset Hierarchy Merging

To merge two subset hierarchies (b-trees), a process takes advantage of the binary structure to boost efficient location of a closest branch for merging. First, the process starts with one of the two trees, denoted as tree 1. For each leaf in tree 1, the process searches the other tree (tree 2) to find the best match in a top-down manner. As illustrated in Figure 2, to find a match for document/leave \( e \) in tree 1, the process starts at the root merge of tree 2 and goes downward. At each merge, the left and right branches are compared to \( e \) and the closer branch is selected as the path. This process continues until it reaches the bottom (a leaf) of the tree where a leaf is identified.

In Figure 2, leaf 7 is identified as the match for \( e \). In this case, the process takes the two leaves (7 and \( e \)) and their siblings (8 and \( d \) in Figure 2), and follows a simple agglomerative process to merge the four under the corresponding branch. After all leaves of tree 1 (e.g., a, b, c, d, and \( e \)) are processed, the two trees are merged. When all \( p/2 \) processes are done, \( p \) trees have been merged into \( p/2 \) trees. The next step will rely on \( p/4 \) processes to merge the \( p/2 \) trees and so forth, until there is only one tree.

If \( n \) documents are equally divided among the hierarchies, it takes \( O(\log \frac{n}{p}) \) time to search for each match. Given \( n/p \) leaves in one tree to be merged into the other, the process of merging two trees has a time complexity of \( O(\frac{n}{p} \log \frac{n}{p}) \). The whole process of merging trees in parallel takes \( \log p \) iterations and has a time complexity of \( O(\frac{n}{p} \log \frac{n}{p} \log p) \).

Parallel Computing in Browser

The proposed distributed HAC method can be operationalized to support large-scale clustering for Scatter/Gather on the web, especially when there are a good number of processes to be parallelized for computing. However, a web site might not have sufficient server resources to perform the parallel processes. For such web sites to offer Scatter/Gather functionality, we propose a simply, highly scalable solution that relies on computing resources on the client (browser) side. Using a scripting language commonly supported by browsers, the proposed architecture utilizes collective computing power of individual visitors (through browser) in the distributed clustering process.

In this work, we implement a distributed HAC experimental system based on Javascript. Data are transferred between the server (master) and processes (browsers) as JSON objects through HTTP. Major distributed clustering steps, including subset HAC construction and tree pair merging, are performed in browsers. We test the proposed method with real data.

EVALUATION

Data Collections

We use a benchmark text clustering collection, namely the WebKB 4 universities data, in the experiments. The
data set contains 8,282 web pages collected in 1997 from computer science departments of various universities, which were manually categorized into seven categories such as student, faculty, and department. This was developed by the WebKB project at CMU (Craven et al., 1998) and has been widely used for text clustering and classification research.

**Experimental Setup**

We develop an experimental clustering algorithm based on an open-source vector clustering module in Javascript (Arthur, 2012). We use the Weka machine learning package to pre-process 8000 documents from WebKB before clustering (Witten and Frank, 2005).

We tokenize documents into single words, remove stop-words, and normalize terms using an iterated Lovins stemmer (Lovins, 1968). A number of top frequent words are selected as features (DF thresholding); 1,000 features are used in main experiments. All documents are normalized to unit vectors.

In HAC clustering, we use euclidean distance and average link. We vary the number of documents \( n \in \{1000, 2000, 4000, 8000\} \) and the number of parallel processes \( p \in \{1, 2, 4, 8\} \) in experiments.

**Evaluation Metrics**

Clustering time in each phase is recorded. We analyze efficiency (clustering time for parallel HAC, merging, and overall) and examine its relation to collection size and the number of parallel processes.

In addition, using categorical labels available in the data as gold standard, we evaluate clustering effectiveness based on several classic metrics, namely, purity, rand index, precision, recall, and \( F_1 \). By assigning each cluster to the most frequent class (label) in it, we can compute purity by:

\[
\text{purity}(C, L) = \frac{1}{N} \sum_i \max_j |c_i \cap l_j| \quad (1)
\]

where \( N \) is the total number of documents. \( C \) is the set of clusters and \( c_i \) is the \( i^{th} \) cluster. \( L \) is the set of labels (classes) where \( l_j \) is the \( j^{th} \) label.

Computing the other metrics such as rand index is by viewing document clustering as a series of decision making (Manning et al., 2008). Given Table 1 which summarizes the numbers of correctly and incorrectly clustered document pairs:

<table>
<thead>
<tr>
<th>Labels</th>
<th>System →</th>
<th>Same Cluster</th>
<th>Different clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same Class</td>
<td>TP: True Positive</td>
<td>FN: False Negative</td>
<td></td>
</tr>
<tr>
<td>Difficult Classes</td>
<td>FP: False Positive</td>
<td>TN: True Negative</td>
<td></td>
</tr>
</tbody>
</table>

![Table 1: Decision table of clustering](image)

Rand Index measures the ratio of correct decisions and is computed by:

\[
RI = \frac{TP + TN}{TP + FP + FN + TN} \quad (2)
\]

Likewise, precision, recall, and \( F_1 \) can be computed by:

\[
P = \frac{TP}{TP + FP} \quad (3)
\]

\[
R = \frac{TP}{TP + FN} \quad (4)
\]

\[
F_1 = \frac{2*P*R}{P + R} \quad (5)
\]

Whereas rand index measures clustering accuracy by taking into account both true positive and true negative, classic IR evaluation metrics such as precision and recall emphasize the ability to find relevant answers/pairs (true positive). Purity and precision are similar in that they both focus on the internal accuracy within each cluster. Recall, on the other hand, addresses the effectiveness of having as many relevant document pairs as possible in one cluster. With these various metrics, we may examine the strength and weakness of the proposed method in multiple perspectives.

**Evaluation Results**

Table 2 shows clustering time for the 1000- and 8000-document collections. With each collection, when the number of processes increases, HAC processing time in the first phase \( \tau_{proc} \) dramatically decreases whereas tree pair merging time \( \tau_{merge} \) remains roughly at the same scale. Merging, relying on efficient b-tree searches, takes much less time than HAC construction does. Overall clustering time primarily consists of HAC construction time in phase one and decreases rapidly with an increase in the number of processes.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( p )</th>
<th>( \tau_{proc} ) (ms)</th>
<th>( \tau_{merge} ) (ms)</th>
<th>( \tau ) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>14,758</td>
<td>0</td>
<td>14,758</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>3,597</td>
<td>220</td>
<td>3,817</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>946</td>
<td>200</td>
<td>1,146</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>232</td>
<td>171</td>
<td>403</td>
</tr>
<tr>
<td>8000</td>
<td>1</td>
<td>946,610</td>
<td>0</td>
<td>946,610</td>
</tr>
<tr>
<td>8000</td>
<td>2</td>
<td>250,805</td>
<td>2,169</td>
<td>257,974</td>
</tr>
<tr>
<td>8000</td>
<td>4</td>
<td>59,913</td>
<td>5,234</td>
<td>65,149</td>
</tr>
<tr>
<td>8000</td>
<td>8</td>
<td>15,215</td>
<td>3,369</td>
<td>18,584</td>
</tr>
</tbody>
</table>

![Table 2: Parallel hierarchical clustering on 1K and 8K document collections](image)

**Efficiency vs. collection size**

Now we look at the influence of the number of documents \( n \) on clustering time given a fixed number of parallel processes \( p \). Table 3 shows example data with \( p = 8 \) whereas Figure 3 plots clustering time vs. the number of documents.

We identify from the data that overall clustering time \( \tau \propto n^{1.85} \) with \( p = 8 \). Likewise, we find that \( \tau \propto n^{2.0} \) with \( p = 2 \) and \( \tau \propto n^{1.94} \) with \( p = 4 \). From \( p = 8 \) to \( p = 2 \), the exponent in the relation decreases.

Having a large number of processes appears to subdue the influence of collection size (a smaller exponent discussed above). This is an important observation. The implication is that a website with a larger user base

![Table 3: Clustering time for fixed number of processes](image)
Table 3: Parallel hierarchical clustering with 8 processors

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>$\tau_{proc}$ (ms)</th>
<th>$\tau_{merge}$ (ms)</th>
<th>$\tau$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>S</td>
<td>232</td>
<td>171</td>
<td>403</td>
</tr>
<tr>
<td>2000</td>
<td>S</td>
<td>926</td>
<td>306</td>
<td>1,232</td>
</tr>
<tr>
<td>4000</td>
<td>S</td>
<td>3,767</td>
<td>732</td>
<td>4,499</td>
</tr>
<tr>
<td>8000</td>
<td>S</td>
<td>15,215</td>
<td>3,369</td>
<td>18,584</td>
</tr>
</tbody>
</table>

Table (traffic) also has the potential capacity to deal with much larger text collections.

Figure 3: Clustering time vs. # documents with 8 parallel processes

Efficient vs. # processes

Figure 4 plots clustering time $\tau$ vs. the number of processes $p$ on the 1000-document collection. As shown in Figure 4 (b), clustering time $\tau$ vs. # processes $p$ is roughly linear on log/log coordinates. We identify from the data points that $\tau \propto 1/p^1.7$.

Figure 4: Clustering time vs. # parallel processes on 1K documents

Experiments on the 8000-document collection shows a similar pattern, with the clustering time's relation to # processes being roughly $\tau \propto 1/p^{1.9}$. These results are consistent with our earlier analysis in which the time complexity of distributed HAC is $O(n^2/p^2)$, which suggests a function of $\tau \propto 1/p^2$. The actual exponents 1.7 and 1.9 are slightly less than 2, likely due to overhead in tree merging, data transfer and initialization.

Again, the influence of # processes $p$ on clustering time $\tau$, given $\tau \approx 1/p^2$, suggests that clustering efficiency (and scalability) can be dramatically improved if a large number of processes are available. With the browser-based distributed architecture, the more traffic a web site has, the more powerful and efficient it will be in performing clustering.

Clustering effectiveness

Table 4 presents preliminary results on clustering effectiveness of a single clustering process vs. distributed clustering of 8 parallel processes (on 8k documents). The single clustering process appears to perform better in terms of precision and recall whereas distributed HAC has higher purity and rand index scores.

Whereas rand index measures clustering accuracy by taking into account both true positive and true negative, classic IR evaluation metrics such as precision and recall emphasize the ability to find relevant answers/pairs (true positive). The parallel method appears to produce good outcomes in terms of both true positives and true negatives. We plan to conduct further experiments and analyses to better understand implications of the proposed distributed HAC method on clustering effectiveness.

SCATTER/GATHER ON THE WEB

Based on the research on large-scale clustering, we have developed several prototype systems for Scatter/Gather browsing on static document collections as well as on dynamic data volumes. Previous systems for browsing benchmark text collections can be found in Ke et al. (2008, 2009). Here we present a new development for interactive clustering and dynamic browsing of web search results. The system is publicly available on-line (Gong et al., 2012).
Figure 6 is a screen shot of the Scatter/Gather interface, which integrates Bing search API and an efficient on-line clustering method. To interact with the system, the user enters a free text query in the search box and hits Search. In response, the system forwards the query to Bing, retrieves hundreds of results, and performs clustering on the search results to identify main themes. Figure 6 shows major clusters of search results based on query “fax machine.” By default, seven clusters will be shown but the user may adjust the number of clusters by moving the “Desired Number of Clusters” slider. Each cluster panel includes information about the number of results, representative keywords, and first two documents in the cluster. The user can click on “More” to see additional results in the cluster and “Less” to hide them. Related searches and search history are shown on the left hand side, which the user can pick to quickly enter a new query.

To the right of each cluster, there is also a check box with which the user can select or deselect the cluster. After selecting clusters she is interested in, the user may click the “Gather and Scatter” button to perform clustering. All web documents in selected clusters will be gathered as one collection and scattered into the desired number of clusters. The user can repeat this process until she reaches a relevant subset. In each iteration, the user may change the desired number of clusters with the slider on the upper right corner.

The user can click on a link in the result to view a web document. Figure 7 shows an example page selected from a cluster related to “fax machine.” We have implemented a feature to solicit user feedback on the relevance and usefulness of the viewed page.

In this Scatter/Gather prototype for search result browsing, clustering is performed on retrieved documents to identify major themes/topics, from which the user can explore to zero in on relevant items. We have also included a feature for the user to step back from a Scatter/Gather iteration. The “Back” button in Figure 6 will be enabled after the user performs the second Scatter/Gather iteration.

By allowing the user to go forward (zoom-in clustering) and backward (zoom out to higher level clusters) in the results, the system can potentially help the user identify relevant information more effectively and efficiently. This provides an alternative to classic search result presentation with a long, sequential list of retrieved documents.
CONCLUSION

The proposed distributed HAC clustering method aims to tackle the efficiency challenge of generating a large-scale hierarchy. Our immediate goal is to provide an efficient, scalable solution as an off-line clustering module in the LAIR2 model for Scatter/Gather on the web. Nonetheless, the distributed process can be applied to other domain areas in which hierarchical clustering on large data is needed.

Experimental evaluation in this study has shown that clustering time is dramatically reduced with an increased number of parallel processes. Clustering time $\tau$ is roughly $\tau \propto 1/p^2$, where $p$ is the number of processes. This shows great potentials in clustering scalability given the browser-based distributed architecture. For a website offering interactive Scatter/Gather clustering, serving more traffics will not degrade clustering performance. On the contrary, by utilizing the collective computing power in visitors’ browsers, clustering efficiency will be significantly improved with a larger user base.

Surely careful examination and tuning are needed to understand how much computing resources on the user’s client/browser computer should be utilized. On the one hand, it is beneficial to a web site to have more computing powers for its overall system efficiency. On the other, a site should not over-utilize it and slow down a client computer. A real-world application of the proposed architecture should take into account various constraints on the browser side.

Based on our efforts to make clustering efficient and scalable, we have developed several Scatter/Gather browser prototypes to demonstrate potential applications of interactive clustering. We present a newly developed system for interactive browsing of web search results.

We plan to further investigate Scatter/Gather effectiveness for information retrieval with related clustering methods. When clustering efficiency is no longer a hurdle, a wide spectrum of Scatter/Gather applications can be built to facilitate searching, browsing, exploration, and learning in information seeking processes.

References


results. In SIGIR ’96, pages 76–84, New York, NY, USA.


