ISSN: 0975-8585

Research Journal of Pharmaceutical, Biological and Chemical Sciences

An Efficient General Decentralaized Clustering using Gossip Based Communication

Janani P[1]*, Shanthakumar P[2]

[1]PG Scholar, M. Kumarasamy College of Engineering, Karur, Tamil Nadu, India.
[2]Professor, Department of Computer Science & Engineering, V.S.B. Engineering College, Karur, Tamil Nadu, India.

ABSTRACT

In many standard applications like peer-to-peer systems, large amounts of data are distributed among multiple sources. Analysis of this data and identifying clusters is difficult due to process, storage, and transmission costs. A GD Cluster, a general fully decentralized clustering method, which is capable of clustering dynamic and distributed data sets. Nodes continuously cooperate through decentralized gossip-based communication to maintain summarized views of the data set. We customize GD Cluster for execution of the partition-based and density-based clustering methods on the summarized views, and also offer enhancements to the basic algorithm. Coping with dynamic data is made possible by gradually adapting the clustering model. We Proposed a Decentralized Clustering Frame Work Search Engines. Coping with dynamic data is made possible by gradually adapting the clustering model. Our experimental evaluations show that GD Cluster can discover the clusters efficiently with scalable transmission cost, and also expose its supremacy in comparison to the popular method LSP2P.

Keywords: Distributed systems, clustering, partition-based clustering, density-based clustering, dynamic system.

Corresponding author

ISSN: 0975-8585

INTRODUCTION

World Wide Web could be a terribly large distributed digital data space. The flexibility to search and retrieve data from the web efficiently and effectively is an enabling technology for realizing its full potential. Current search tools retrieve too several documents, of which only a small fraction are relevant to the user query. Furthermore, the most relevant documents do not necessarily appear at the top of the query output order. Clustering Techniques are now being used to provide a meaningful search result on web. Text document clustering has been traditionally investigated as a means of up the performance of search engines. We presenting a thorough comparison of the algorithms based on the assorted aspects of their features and functionality. Furthermore, we highlight the main characteristics of a number of existing web clustering engines and also discuss how to evaluate their retrieval performance is Low.

With the increase in information on the World Wide Web it has become difficult to find the desired information on search engines. One approach that tries to solve this problem is using clustering techniques for grouping similar documents together in order to facilitate presentation of results in more compact form and enable thematic browsing of the results set. It is used to give a meaningful search result on web. The four main criteria for creating cluster categories: Making the titles concise, accurate, distinctive, and "humanlike" -- in other words, not something that looks like it was generated by a machine. One common feature of most current common approach in distributed cluster is to mix and merge native representations in an exceedingly central node, or aggregate native models in an exceedingly hierarchical data structure [2], [3]. Some recent proposals, though being fully decentralized , include synchronization at the end of each round, and/or need nodes to keep up history of the cluster [4], [5], [6], [7]. In this paper, a general distributed cluster algorithm (GD Cluster) is projected and instantiated with two standard partition-based and density-based cluster strategies.

We initial introduce a basic methodology during which nodes bit by bit build a summarized read of the data set by endlessly exchanging data on data things and data representatives victimization gossip-based communication. Gossiping [8] is employed as a simple, robust and efficient dissemination technique , that assumes no predefined structure in the network. The summarized read may be a basis for execution weighted versions of the cluster algorithms to produce approximations of the ultimate cluster results.

GD Cluster can cluster a data set that is distributed among a large number of nodes in a distributed atmosphere. It will handle 2 categories of cluster, namely partition-based and density-based, whereas being totally redistributed, asynchronous, and conjointly adjustable to churn. The general design principles utilized within the proposed algorithm also enable customization for different categories of cluster, which are ignored of this paper. we tend to conjointly discuss enhancements to the algorithm significantly geared toward rising communication costs.

The simulation results given victimization real and artificial data sets, show that GD Cluster is ready to realize a high quality global cluster solution, which approximates centralized cluster. we tend to conjointly justify effects of assorted parameters on the accuracy and overhead of the algorithm. We compare our proposal with central cluster and with the LSP2P algorithm [4], and conjointly show its ascendancy in clustering engines is that they do not maintain their own index of documents; similar to meta search engines, they take the search results from one or more publicly accessible search engines. The low precision of the web search engines coupled with the long ranked list presentation make it hard for users to find the information they are looking for. It takes lot of time to find the relevant information. Typical queries retrieve hundreds of documents, most of which have no relation with what the user was looking for. According to this, we considered Web-snippet clustering engine is a useful complement to the flat, ranked list of results offered by classical search engines (like Google).

ISSN: 0975-8585

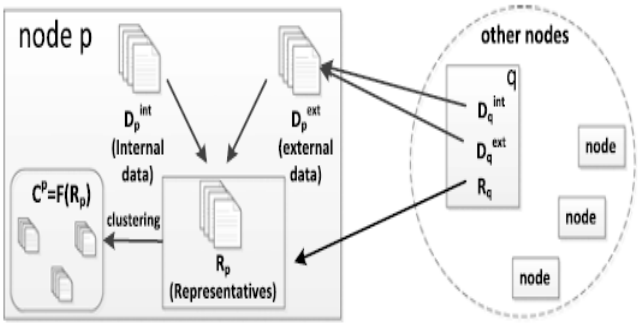


Fig. 1. A graphical view of the system model.

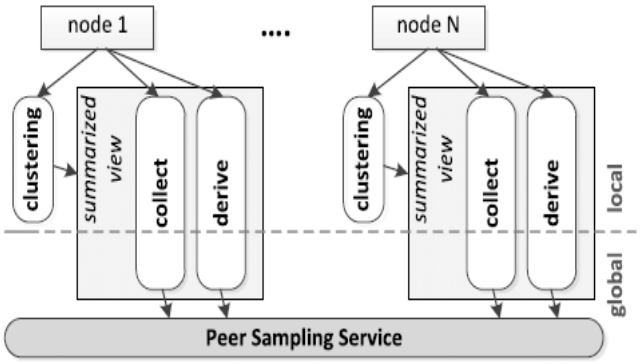


Fig. 2. The overall view of the algorithm tasks.

Web snippet (short description) clustering also known as Web Search Result Clustering is an attempt to apply the idea of clustering to snippets returned by a search engine in response to a query. Thus, it can be perceived as a way of organizing the snippets into set of meaningful thematic groups. Actually, clustering engines are usually seen as complementary instead of alternative to search engines. Not only has search results clustering attracted considerable commercial interest, but it is also an active research area, with a large number of published papers discussing specific issues and systems.

Search results clustering is clearly related to the field of document clustering but it poses unique challenges concerning both the effectiveness and the efficiency of the underlying algorithms that cannot be addressed by conventional techniques. The main difference is the emphasis on the quality of cluster labels, whereas this issue was of somewhat lesser importance in earlier research on document clustering. A clustering engine tries to address the limitations of current search engines by providing clustered results as an added feature to their standard user interface and meaningful labels. This paper gives an idea about document clustering, Web Page document clustering and clustering engines.

SYSTEM MODEL

We consider a group P ¼ fp1; p2; . . . ; png of n networked nodes. Each node p stores and shares a group of data items means p , denoted as its internal data, which can modification over time. D ¼ S p2Pdint p is the set of all data items obtainable within the network. every data item d is conferred victimization AN attribute vector denoted as data. Whenever transmission of data things is mentioned within the text, transmission of the respective attribute vector is meant.

While discovering clusters, p may additionally store attribute vectors of data items from alternative nodes. these things are referred to because the external data of p, and denoted as Dext p . The union of internal and external data items of p is observed as Dp ¼ Dint p [Dext p].

ISSN: 0975-8585

During algorithm execution, every node p gradually builds a summarized view of D, by maintaining representatives, denoted as Rp ¼ frp 1; rp 2; . . . ; rp kpg. each representative r two Rp is a synthetic data item, summarizing a subset Dr of D. The attribute vector of r, rattr, is ideally the average of attribute vectors1 of data items in Dr. The intersection of these subsets need not be empty, i.e., 8r; r0 2 Rp:jDr \ Dr0j 0. the particular set Dr isn't maintained by the algorithm, and is discarded once r is produced. Each data item or representative x in p, has an associated weight wpðxÞ. the load of x is capable the amount of data items that, p believes, x consists of. counting on whether x may be a representative or a data item, wpðxÞ ought to ideally be adequate jDxj or one, severally.

The goal of this work is to create certain that the whole data set is clustered in a very absolutely decentralized fashion, such that each node p obtains AN accurate clustering model, without collecting the complete data set. The illustration of the clustering model depends on the particular clustering method. For partition-based and density-based clustering, a centroid and a collection of core points will function cluster indicators, respectively. Whenever the actual form of clustering is not necessary, we tend to discuss with the clustering methodology simply as F. Fig. 1 provides a summarized read of the system model.

DECENTRALIZED CLUSTERING

Each node gradually builds a summarized read of D, on which it will execute the clustering algorithm F. within the next sections, we tend to initial discuss however the summarized read is constructed. Afterwards, the strategy of weight calculation is represented, followed by the execution procedure of the clustering algorithm.

Building the Summarized read

As represented in Section two, we tend to assume that the whole data set can be summarized in every node p, by means that of representatives. Each node p is accountable for derivation accurate representatives for a part of the data set situated close to means p . For other parts, it exclusively collects representatives. consequently, it gradually builds a global view of D. each node endlessly performs 2 tasks in parallel:

(i) Representative derivation, which we tend to name DERIVE and ii) representative collection, which we tend to name COLLECT. the two tasks can execute repeatedly and endlessly in parallel. a top level view of the tasks performed by every node is demonstrated in Fig. 2. We use two gossip-based, decentralized cyclic algorithms to accomplish the two tasks, as represented within the next sections.

DERIVE

To derive representatives for a part of the information set situated near Dint p , p should have an correct and up-to-date view of the data situated around every data d a pair of means p In every spherical of the DERIVE task, every node p selects another node q for a three-way data exchange, as shown in Fig. 3. It should initial send dint p to node q. If size of dint p is large, it can summarize the internal data by an arbitrary technique corresponding to grouping the data using clustering, and causing one data from each group. Node p then receives from q, data item located in radius r of each d two means p , based on a distance function d. r could be a user-defined threshold, which might be adjusted as p continues to discover data (to that we have a tendency to come to in Section 6.1). within the same manner, it'll additionally send to Q the data in dp that lie at intervals the r radius of data in means q

ISSN: 0975-8585

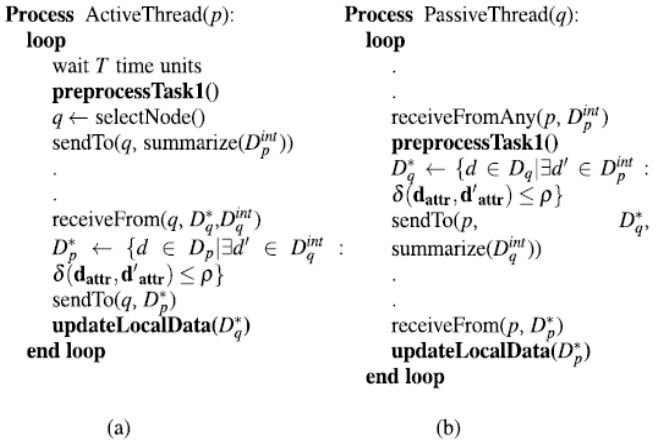


Fig. 3. Task DERIVE: (a) active thread for p and (b)passive thread for selected node q.

.

The operation updateLocalDataðÞ is employed to feature the received data to Dext p . Knowing some data situated at intervals radius r of some internal data item d, node p will summarize all this data into one representative. this can be performed periodically each t gossip rounds using the algorithm of Fig. 4. The merge Weights operate, updates the representative weight, and is later represented in Section three.3

COLLECT

To fulfill the COLLECT task, every node p selects a random node each T time units, to exchange their set of representatives with one another (Fig. 5). each nodes store the total set of representatives. The summarize operate utilized in the algorithm, simply returns all the representatives given to that as input. A special implementation of this perform is represented in Section 5.1, which reduces the quantity of representatives. Initially, every node has solely a group of internal data items, Dint p. Thus, the set of representatives at each node is initialized with all of its data things, i.e., Rp ¼ Dint p.

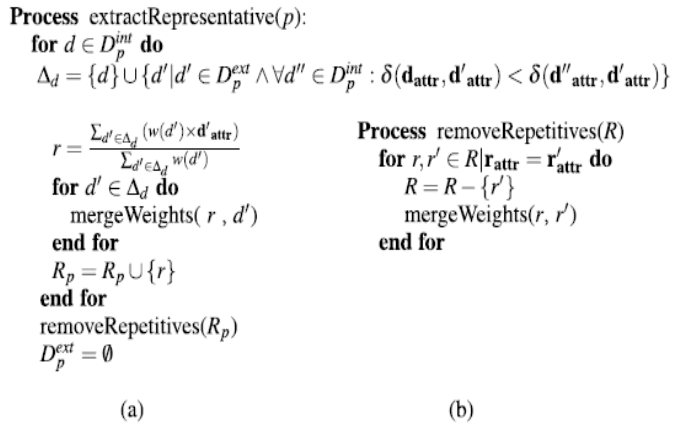


Fig. 4. (a) Extracting representatives from the collected data, (b) removing repetitive representatives.

ISSN: 0975-8585

```
Process ActiveThread(p):
  loop
    wait T time units
    preprocessTask2()
    q ← selectNode()
    sendTo(q, summarize(R_p, |R_p|))
    .
    .
    receiveFrom(q, R*_q)
    R_p = R_p ∪ R*_q
    removeRepetitives(R_p)
  end loop
```

(a)

```
Process PassiveThread(q):
  loop
    .
    .
    .
    .
    receiveFromAny(p, R*_p)
    preprocessTask2()
    sendTo(p,
    summarize(R_q,|R_q|))
    R_q = R_q ∪ R*_p
    removeRepetitives(R_q)
  end loop
```

(b)

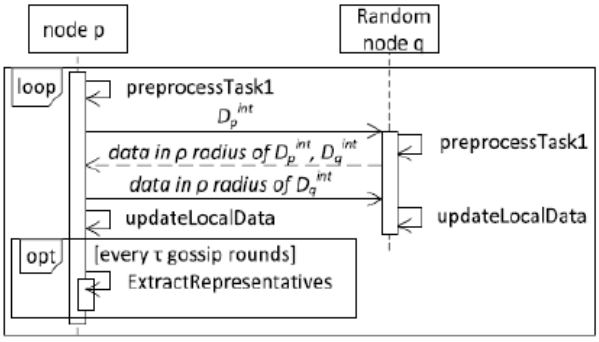
Fig. 5. Task COLLECT: (a) active thread for p and (b) passive thread for selected node q.

The two algorithms of tasks DERIVE and COLLECT, start with a preprocessing operation. in this basic algorithm, these operations haven't any special operate, so we defer their discussion to Section 4. The graphical illustration of the communication performed in DERIVE and COLLECT is represented in Fig. 6. The operation select Node Þ employed in Figs. 3 and 5, employs a peer-sampling service to come a node elite uniformly arbitrarily from all live nodes within the system (see, e.g., [9]).

Diffusion Speed

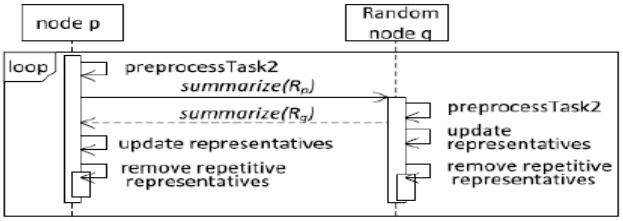
In tasks DERIVE and COLLECT we use gossiping as a propagation media. this can be particularly completely different from aggregation protocols [8] that use gossiping to achieve agreement on aggregations. using vocabulary of [8] and ignoring the details, the general approach of GD Cluster is simplified as follows. at all times t, a node p maintains an ordered set (not a sum) stp, initialized to s0;p ¼ dint p , and an ordered set of corresponding weights wt p. At every time step t, p chooses a target node ftðpÞ uniformly arbitrarily and sends each collections to that node and itself. It calculates union of the received pairs ð^sr; ^wrÞ from different nodes with its own s and w sets. In step t of the algorithm, st;p is that the view p has on the entire dataset, whereas wt;p contains the corresponding weight of each view component. because the set s quickly becomes large, the notion of representatives are introduced. Node p can summarize the elements of st;p by removing a subset, computing the average of its components (locally), and replacing the typical value in st;p. The corresponding weights ought to even be removed and replaced by the aggregate weight. This summarized view is tagged Rp during this paper.

According to [8] and [10], a message that originate with pat time t0 and is forwarded by all nodes that have received it, can reach all nodes in time at the most 4 log N þ log 2 d with probability a minimum of one d 2. Therefore, once a similar time order, the summarized view of p, can have parts from all alternative nodes, either in their raw kind or embedded in a very representative.



(a)

ISSN: 0975-8585



(b)

Fig.6. A schematic diagram of Weight Calculation

Weight Calculation

When representatives are incorporate, as an instance in the perform Remove Repetitive, a special technique ought to be devised for weight calculation. The algorithm doesn't record the set Dr for every representative r, because of resource constraints. Also, there's a clear stage of intersection between summarized data of various representatives. to handle the load calculation issue, representative points are among a (small size) "estimation field", that enables us to approximate the number of actual things it represents.

Final clustering

The final clustering algorithm F is executed on the set of representatives in a very node. Node p will execute a weighted version of the clustering algorithm on Rp, any time it needs, to realize the ultimate clustering result. in a very static setting, continuous execution of DERIVE and COLLECT can improve the standard of representatives inflicting the clustering accuracy to converge. in the following, we tend to discuss partition-based and density-based clustering algorithms as examples.

Partition-based clustering

K-means [12] considers data items to be placed in an m-dimensional topological space, with an associated distance measured. It partitions the data set into k clusters, C1,C2, . . . , Ck. each cluster Cj contains a centroid mj, that is outlined as the average of all data assigned to that cluster. This algorithm tries to reduce the subsequent objective function:

$$K \Sigma \Sigma || dl-\mu j ||2 \quad \text{eq. (1)}$$
$$J=1\ dl€\ cj$$

Weighted K-means assumes a positive weight for each data item and uses weighted averaging. They themselves will be assigned weight values, indicating variety of data appointed to the clusters. The formal definition of the weighted Kmeans is given thoroughly.

The algorithm proceeds heuristically. a set of random centroids are picked at first, to be optimized in later iterations. The obtainable approaches of distributed partition-based clustering usually assume identical initial K-means centroids all told nodes [4], [5], [6]. This is, however, not needed in our algorithm as every node will use an arbitrary parameter k with an impulsive set of initial centroids.

Density-Based clustering

In density-based clustering, a node p can execute, for instance, a weighted version of DBSCAN on Rp with parameters minPts and ". In DBSCAN, item is marked as a core point if it has at least minPts data items at intervals its radius.

Also, 2 core points are among one cluster, if they are in "range of each other, or are connected by a sequence of core points, where each two consecutive core points have a maximum distance of ". A non-core

data item set at intervals distance from a core purpose, is within the same cluster as that core purpose, otherwise it's an outlier.

In our algorithm, each representative may cover a locality with radius a pair of bigger than. conjointly a representative doesn't necessarily have constant attribute vector as any regular data item. Therefore, representatives don't directly mimic core points. all the same, core points in DBSCAN are a way of describing data density. Adhering to the current thought, representatives may indicate dense areas.

The ᵖ parameter of the DERIVE task may be set to €. This ensures that if some data item could be a core point, the corresponding derived representative can have a minimum weight of minPts. This customization conjointly suggests that per each internal data item, at the most minPts data items ought to be transferred in COLLECT. one amongst the advantages of DBSCAN is its ability to discover outliers. to realize this in our algorithm, task COLLECT ought to be customised to transfer solely representatives with weight larger than minPts. This causes representatives settled outside the particular clusters to not be disseminated within the network, and improves the general clustering accuracy. The density-based clustering methodology simply described may be thought of a rather changed version of the distributed density-based clustering algorithm Go Scan.

In Go Scan nodes detect core points and disseminate the through methods very similar to COLLECT and DERIVE. Go Scan is an exact method, whereas here we are providing an approximate method. The approximation imposes less communication overhead, and faster convergence of the algorithm.

Dynamic *Data Set:*

Real-world distributed systems change continuously, because of nodes joining and leaving the system, or because Their set of internal data is modified. To model staleness of data, each data item will have an associated age. Agep(d denotes the time that node p believes has passed since d was obtained from its originating, owning node. Time is measured in terms of gossiped rounds. The age of data items accompany them in the DERIVE task. The age of an external data item at node p is increased (by p) before each communication; the age of an internal data always remains zero to reflect that it is stored (and upto-date) at its owner. If a node p receives a copy d0 of a data item d it already stores, age p(d) is set to min{age p(d),age p(d')} (and d'is further ignored).

When a data item d is removed from the original peer, the minimal recorded age among all its copies will only increase. Node p can remove data item d if age p(d) > MaxAge, where MaxAge is some threshold value, presuming that the original data item has been removed. An age argument is also associated with each representative; age p(r) is set to zero when r is first produced by p, and increased by one before each communication.

The weight of a data item or a representative is a function of its age. For a data item d, the weight function is ideally one for all age values not greater than MaxAge. The data items summarized by a representative have different lifetimes according to their age. Therefore, the weight of the representative should capture the number of data items summarized by the representative at each age value. When the weight value falls to zero, the representative can be safely removed.

We will see below that instead of the actual weight, the weight estimators are stored per each age value to enable

Further merging and updating of representatives. The weight function of a representative will always be in the form of a descending step function for values greater than age p(r), and will reach zero at most at age p(r)+ MaxAge. All of the data currently embedded in the representative will be gradually removed, and no data can last longer that MaxAge units from the current time. With the weight function being dependent on age, the weight estimators are in turn bound to the age values. ^wpl(x, t) presents the l'th weight estimator of item x in age t, from the view of peer p. For a data item d, while age p(d) ≤ MaxAge, each weight estimator preserves its initial value, and is null otherwise. For a representative r, s weight estimators are recorded at each age value greater than age p(d), up to the point where all data embedded in the representative are

removed. To incorporate these new concepts in the basic algorithm, the two preprocessing operations of DERIVE and COLLECT should be modified to increase age values of data and representatives, and remove them if necessary. Moreover, before storing the received data in DERIVE, the age values for repetitive data items should be corrected.

Enhancements:

In this section we discuss a number of improvements to the basic algorithm, to enhance the consumed resources.

Summarization

Nodes may have limited storage, processing and communication resources. The number of representatives maintained at a node increases as the DERIVE and COLLECT tasks proceed. When the number of representatives and external data items stored at p exceeds its local capacity LCp, first the representative extraction algorithm of Fig. 4 is executed to process and then discard external data. Afterwards, the summarization task of Fig. 11 is executed with parameters Rp and αLCp, and the result is stored as the new Rp set. $0 < \alpha < 1$ is a locally determined parameter, controlling consumption of local resources. Dealing with limitations of processing resources is similar. If the number of representatives and external data items to be sent by p in the DERIVE and COLLECT tasks, exceeds its communication capacity CCp, the same summarization task is executed with parameters Rp and βCCp. Thus, a reduced set of representatives is obtained. $0 < \beta < 1$ is a parameter controlling the number of transmitted representatives. If the external data items to be sent in the DERIVE task exceed the communication limits, sampling is used to reduce the amount of data. The summarization task actually makes use of weighted K-means (described in Section 3.3.1), which
effectively "summarizes" a collection of data items by means of a single representative with an associated weight.

Performance Evaluation:

We evaluate the GD Cluster algorithm in static and dynamic settings. We will also compare GD Cluster with a central approach and with LSP2P, a recently proposed algorithm being able to execute in similar distributed settings.

Evaluation Model

We consider a system of N nodes, each node initially holding a number of data items, and carrying out the DERIVE and COLLECT tasks iteratively. For simplicity and better understanding of the algorithm, we consider only data churn in the dynamic setting. In each round, a fraction of randomly selected data items is replaced with new data items. By using the peer sampling service, the network structure is not a concern in the evaluations. Each cluster in the synthetic data sets consists of a skewed set of data composed from two Gaussian distribution with different values of mean and standard deviation. The real data sets used for the partition-based clustering are the well-known Shuttle, MAGIC Gamma Telescope, and Pen digits data sets 3. These data sets contain nine, 10, and 16 attributes, and are clustered into seven, two, and 10 clusters, respectively. From each data set, a random sample of 10, 240 instances are used in the experiments. To assign the data set D to nodes, two data-assignment strategies are employed, which aid at revealing special behaviors of the algorithm:

1. Random data assignment (RA): Each node is assigned data randomly chosen from D.
2. Cluster-aware data assignment (CA): Each node is assigned data from a limited number of clusters.

The second assignment strategy abates the average number of nodes that have data close to each other. Such a condition reduces the number of other nodes which have target data for the COLLECT task. When applying churn, in the first assignment strategy, data items are replaced with random unassigned data items. The second data assignment strategy allows concept drift when applying churn, by reserving some of the clusters and selecting new points from these clusters. Concept drift refers to change in statistical properties of the target data set which should be clustered. Nodes can adjust the r parameter during execution

ISSN: 0975-8585

based on the incurred communication complexity. In the evaluations, for simplicity, the r parameter is selected such that the average number of data located within the r radius of each data item is equal to 5. Different parameters used in conducting the experiments, along with their value ranges and defaults, are presented in Table 1. The parameter values are selected such that special behaviours of the algorithm are revealed. LC and CC are measured as multiples of the required resource for one representative. The majority of the evaluations are performed with partition- based clustering. Partial evaluation on density based clustering is discussed at the end of the section.

Table 1: Evaluation modeling

Symbol	Description	Range (default)
N	Number of nodes	128-16,384 (128)
|C|	Number of real clusters in the data set	8-50
Nint	Number of internal data items per node	2-1,000(10)
S	Number of weight estimators	20
T	The period between representative extraction in DERIVE	<0.4
churn	Fraction of data replaced in each gossip	10-50%
ratio	round	(10%)
MaxAge	Threshold for the age parameter	2-38(10)
LC	Node storage capacity	20-1,280 (100)
α	The parameter used in summarizing local representatives	0.5
CC	Node communication capacity	<3|C|
β	The parameter used in summarizing communicated representatives	<0.5

4. SEARCHING METHODOLOGIES

There are two types of search algorithms: algorithms that don't make any assumptions about the order of the list, and algorithms that assume the list is already in order. We'll look at the former first, derive the number of comparisons required for this algorithm, and then look at an example of the latter. In the discussion that follows, we use the term search term to indicate the item for which we are searching. We assume the list to search is an array of integers, although these algorithms will work just as well on any other primitive data type (doubles, characters, etc.). We refer to the array elements as items and the array as a list.

Linear Search:

The simplest search algorithm is linear search. In linear search, we look at each item in the list in turn, quitting once we find an item that matches the search term or once we've reached the end of the list. Our "return value" is the index at which the search term was found, or some indicator that the search term was not found in the list.

Algorithm for linear search

```
For (each item in list)
{
Compare search term to current item
If match,
Save index of matching item
break
}
Return index of matching item, or -1 if item not found
```

Performance of linear search

When comparing search algorithms, we only look at the number of comparisons, since we don't swap any Values While searching. Often, when comparing performance, we look at three cases:

- **Best case:** What is the fewest number of comparisons necessary to find an item.
- **Worst case:** What is the most number of comparisons necessary to find an item

- **Average case:** On average, how many comparisons does it take to find an item in the list. For linear search, our cases look like this:
- **Best case:** The best case occurs when the search term is in the first slot in the array. The number of Comparisons in this case is 1.
- **Worst case:** The worst case occurs when the search term is in the last slot in the array, or is not in the Array. The number of comparisons in this case is equal to the size of the array. If our array has N items, then it takes N Comparisons in the worst case.
- **Average case:** On average, the search term will be somewhere in the middle of the array. The number of comparisons in this case is approximately N/2.

In both the worst case and the average case, the number of comparisons is proportional to the number of items in the array, N. Thus, we say in these two cases that the number of comparisons is order N, or O (N) for short. For the best case, we say the number of comparisons is order 1, or O (1) for short.

Binary Search:

Linear search works well in many cases, particularly if we don't know if our list is in order. Its one drawback is that it can be slow. If N, the number of items in our list, is 1,000,000, then it can take a long time on average to find the search term in the list (on average, it will take 500,000 comparisons). What if our list is already in order. Think about looking up a name in the phone book. The names in the phone book are ordered alphabetically. Does it make sense, then, to look for "Sanjay Kumar" by starting at the beginning and looking at each name in turn? No! It makes more sense to exploit the ordering of the names, start our search somewhere near the K's, and refine the search from there. Binary search exploits the ordering of a list. The idea behind binary search is that each time we make a comparison; we eliminate half of the list, until we either find the search term or determine that the term is not in the list. We do this by looking at the middle item in the list, and determining if our search term is higher or lower than the middle item. If it's lower, we eliminate the upper half of the list and repeat our search starting at the point halfway between the first item and the middle item. If it's higher, we eliminate the lower half of the list and repeat our search starting at the point halfway between the middle item and the last item.

Algorithm for binary search

```
set first = 1, last = N, mid = N/2
while (item not found and first < last)
{
compare search term to item at mid
if match
save index
break
else if search term is less than item at mid,
set last = mid-1
else
set first = mid+1
set mid = (first+last)
}
return index of matching item, or -1 if not found
```

Performance of binary search

The best case for binary search still occurs when we find the search term on the first try. In this case, the search term would be in the middle of the list. As with linear search, the best case for binary search is O(1), since it takes exactly one comparison to find the search term in the list.

The worst case for binary search occurs when the search term is not in the list, or when the search term is one item away from the middle of the list, or when the search term is the first or last item in the list. How many comparisons does the worst case take. To determine this, let's look at a few examples.

ISSN: 0975-8585

Suppose we have a list of four integers: {1, 4, 5, 6}. We want to find 2 in the list. According to the algorithm, we start at the second item in the list, which is 4.2 Our search term, 2, is less than 4, so we throw out the last three items in the list and concentrate our search on the first item on the list, 1. We compare 2 to 1, and find that 2 is greater than 1. At this point, there are no more items left to search, so we determine that 2 is not in the list. It took two comparisons to determine that 2 is not in the list. Now suppose we have a list of 8 integers: {1, 4, 5, 6, 9, 12, 14, 16}. We want to find 9 in the list. Again, we find the item at the midpoint of the list, which is 6. We compare 6 to 9, find that 9 is greater than 6, and thus concentrate our search on the upper half of the list: {9, 12, 14, 16}.

We find the new midpoint item, 12, and compare 12 to 9. 9 is less than 12, so we concentrate our search on the lower half of this list (9). Finally, we compare 9 to 9, find that they are equal, and thus have found our search term at index 4 in the list. It took three comparisons to find the search term. If we look at a list that has 16 items, or 32 items, we find that in the worst case it takes 4 and 5 comparisons, respectively, to either find the search term or determine that the search term is not in the list. In all of these examples, the number of comparisons is log2 N3. This is much less than the number of comparisons required in the worst case for linear search! In general, the worst case for binary search is order log N, or O(logN). The average case occurs when the search term is anywhere else in the list. The number of comparisons is roughly the same as for the worst case, so it also is O(logN).

In general, anytime an algorithm involves dividing a list in half, the number of operations is O(log N).

SIMULATION RESULTS

We start by presenting the simulation results for the static network, and then proceed to dynamic configurations. Evaluation of different parameters is mainly performed with the synthetic data set, as we can efficiently control the number of clusters, data density and the churn ratio.

Static Settings

When network data is persistent, each node gradually learns the data through its representatives, and the clustering accuracy converges. The algorithm behavior in a static setting is shown in Fig.3, where the number of internal data items of each node, N int , varies from 2 to 10. The trend of clustering accuracy convergence against simulation rounds, is shown for basic and enhanced GD Cluster. Convergence is identified by three rounds of minor (less than 1 percent) change in results. The accuracy converges in to 100% and more than 95% in basic and enhanced GD Cluster respectively. The enhanced GD Cluster offers less converged accuracy values due to limited transmission of representatives and data, which reduces the quality of the constructed view of data in each node. As observed, in this setting, when nodes have few data (e.g., Nint = 2), detecting accurate clusters is harder, due to sparseness of clusters.

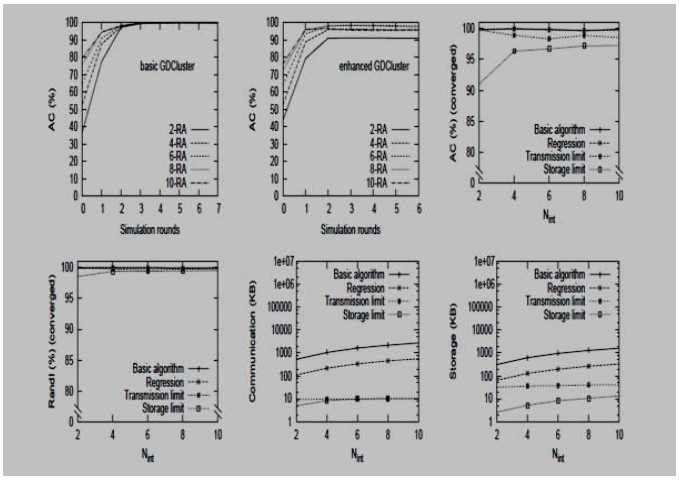


Fig. 7. Convergence and cost evaluation in static settings when Nint varies. Comparing incremental configurations: basic; regression; reduced communication; reduced storage (enhanced GD Cluster).

ISSN: 0975-8585

The same figure compares the basic GD Cluster with three improved versions, when N int varies. Communication and storage overheads show average per round values until convergence for each node. The values are considerable for the basic GD Cluster due to the storage and transmission of a large number of external data items and representatives. The first improved version involves regression to reduce the weight estimators. As expected, this improvement preserves the clustering accuracy, while reducing the resource consumption up to 80%. In the next improvement, the communication capacity is restricted. In this setting, the AC values decrease by approximately 2 percent, while the communication overhead experiences a major reduction. Further limitation of storage capacity in the last improvement, still keeps AC above 95%, but dis allocates local resources.

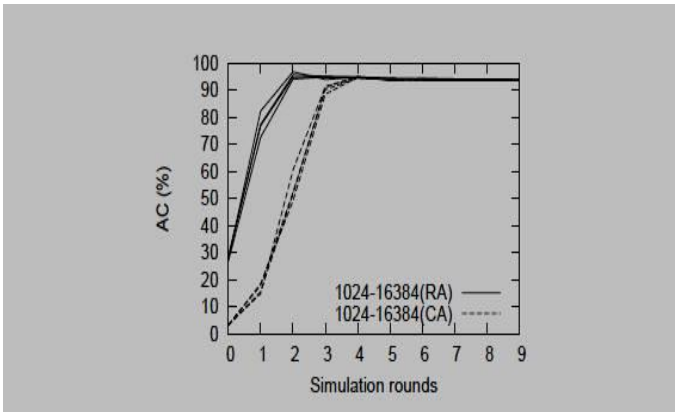


Fig. 8: It shows the behaviour of GDCluster when the network size varies from 1024 to 16384 nodes. The AC values have converged to more than 90%. This shows the efficiency and scalability of the algorithm. In the random data-assignment strategy, AC values are initially higher. This is due to each node initially having internal data items from different clusters, enabling it to identify more clusters. As the performance of the algorithm for different network sizes is very similar, we used the average values of different metrics for *N* = 1024 as a baseline in table 2, and showed the difference of values for other network sizes. Convergence in a static setting, when *N* varies (average values in table 2)

The RandI values converge to 100%. The communication and storage overheads of the algorithm remain constant due to restricting resource consumption. As observed, the differences of values for different network sizes are small, showing scalability of the algorithm. In the evaluation of the algorithm using real data sets, both central K means and GD Cluster are evaluated against the actual labels of data, and the results are presented in Table 3. GD Cluster is executed in a network of 1024 nodes, each having 10 data items. The AC and RandI values for GD Cluster are very close to those of the central K-means. Because GD Cluster executes K-means on the representatives instead of data, when compared to actual data labels, its accuracy may even surpass the central results for some data sets. The results show the efficiency of the algorithm in conforming to central clustering for real-world data.

Dynamic Settings

The MaxAge parameter puts an upper bound on the storage period of external data items, and representatives. Fig.5 shows the evaluation of the basic GDCluster algorithm when MaxAge varies. Very low values of MaxAge prohibit complete propagation of information in the network, and also cause early removal of data and representatives. Large values, on the other hand, maintain invalid information longer than required and degrade accuracy. The optimum behavior of the algorithm is observed when MaxAge is equal to 6.

ISSN: 0975-8585

Table 2: Performance differences when N varies with respect to N = 1024

	Network size	1024 (baseline)	2048-16384
RA	AC (%)	93.85	<0.11
	corrected RandI (%)	100	<0.0011
	communication (KB)	25.44	<0.67
	storage (KB)	13.4	<0.028
CA	AC (%)	93.77	<0.019
	corrected RandI (%)	100	0
	communication (KB))	25.45	<0.31
	storage (KB)	13.33	<0.12

Table 3: Average values of evaluation metrics for 50 runs of the algorithm with real data sets

	AC		RandI		Comm. overhead
Data set	Central Kmeans	GDCluster	Central Kmeans	GDCluster	GDCluster
Shuttle	0.84	0.86	0.75	0.80	16.8
MAGIC	0.68	0.67	0.56	0.56	17.17
Pendigits	0.63	0.53	0.80	0.88	20.39

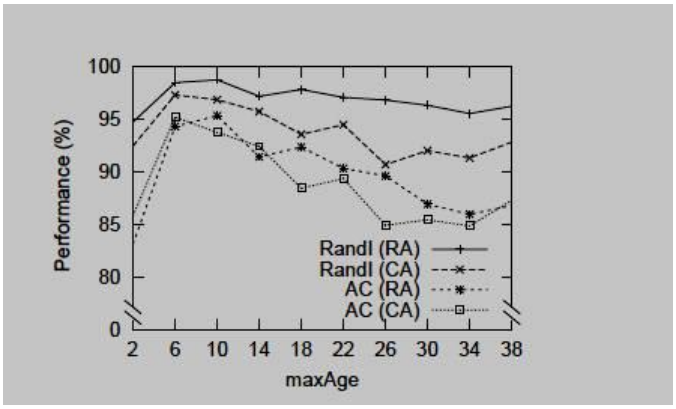


Fig. 9. Effect of changing MaxAge

This is consistent with the earlier observation of quick convergence of the algorithm. Therefore, *MaxAge* should be chosen to be compatible with algorithm convergence rate, as to remove the data at a reasonable pace. Fig. 6 shows the evaluation of the algorithm against different metrics in a dynamic setting, with 10% churn. With the CA strategy, concept drift is observed as some clusters are introduced later to the network. As illustrated in Fig. 6, for all network sizes, the AC value rises to approximate average values of 94% and 93% with the RA and CA strategies, respectively. Although data changes regularly, the RA strategy ensures that previously discovered clusters remain valid through data change. This ensures higher AC values

ISSN: 0975-8585

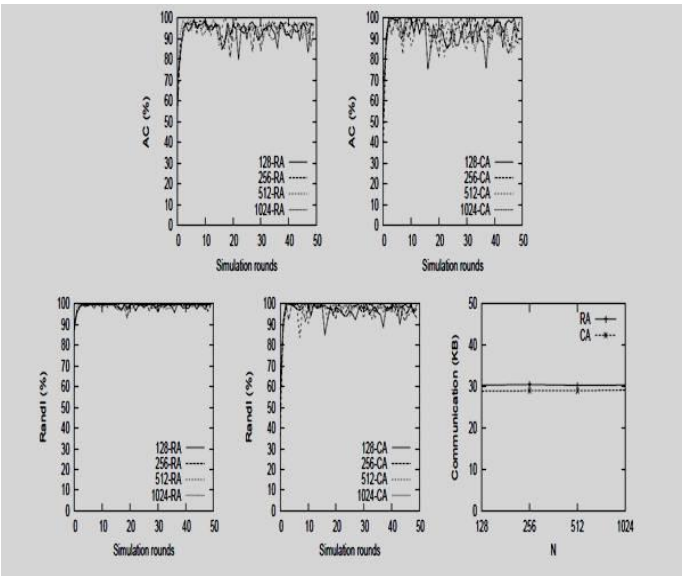


Fig. 10 . Evaluation of GD Cluster in dynamic setting, when N varies.

With concept drift, nodes should move on to discover representatives in the new clusters. It also takes some time for the removed data to be discarded by the embedding representatives. Similar trends are observed for the RandI metric, where approximate average values of 98% and 96% are achieved for RA and CA strategies, respectively. The algorithm has acceptable performance in detecting clusters, even in dynamic settings. Finally the same figure shows that the communication overhead for different network sizes remains roughly the same. This is mainly due to removal of representatives in the dynamic setting which reduces the amount of transferred data between nodes.

Comparison with LSP2P

The LSP2P algorithm executes the K-means in an iterative manner, with each node synchronizing with its neighbors during each iteration. In a static setting, the algorithm is initiated at a single node *p*, which picks a set of random initial centroids along with a termination threshold g > 0 (which we explain shortly). *P* sends these to all its immediate neighbors, and begins iteration 1. When a node receives the initial centroids and threshold for the first time, it forwards them to its remaining neighbors and initiates iteration 1. In each iteration, every node *p* executes one round of K-means on its local data based on the centroids computed inthe previous iteration. It then prompts its immediate neighbors for their corresponding cluster centroids, and updates local centroids based on the received information. Once the computed centroids of two consecutive iterations, deviate less than g from each other, *p* enters the terminated state. In a dynamic setting, the change of data may reactivate the nodes. Regarding the above descriptions on LSP2P, it is observed that the initial centroids are identical in all nodes, which prohibits changing the number of produced clusters.

Also, if K-means is to be executed with different initial centroids, a new instance ofLSP2P should be started. Moreover, the history of executing the K means algorithm is particularly important, and maintained in each node.

CONCLUSION AND FUTURE WORK

In this paper we first identified the necessity of an effective and efficient distributed clustering algorithm. Dynamic nature of data demands a continuously running algorithm which can update the clustering model efficiently, and at a reasonable pace.

We introduced GD Cluster, a general fully decentralized clustering algorithm, and instantiated it for partition-based and density-based clustering methods. The proposed algorithm enabled nodes to gradually build a summarized view on the global data set, and execute weighted clustering algorithms to build the clustering models. Adaptability to dynamics of the data set was made possible by introducing an age factor

which assisted in detecting data set changes updating the clustering model. Our experimental evaluation and comparison showed that the algorithm allows effective clustering with efficient transmission costs, while being scalable and efficient. GD Cluster can be customized for other clustering types, such as hierarchical or grid-based clustering. To accomplish this, representatives can be organized into a hierarchy, or carry statistics of approximate grid cells. Further discussion of these algorithms is deferred to future work.

REFERENCES

[1] K. M. Hammouda and M. S. Kamel, "Hierarchically distributed peer-to-peer document clustering and cluster summarization," IEEE Trans. Knowl. Data Eng., vol. 21, no. 5, pp. 681–698, May 2009.

[2] Y. Pei and O. Za€ıane, "A synthetic data generator for clustering and outlier analysis," Dept. Comput. Sci., Univ. Alberta, Edmonton, AB,Canada, Tech. Rep. TR06-15, 2006.

[3] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, "The sequoia 2000 storage benchmark," in ACM SIGMOD Rec., vol. 22, no. 2, pp. 2–11, 1993.

[4] N. Visalakshi and K. Thangavel, "Distributed data clustering: A comparative analysis," in Foundations Computational Intelligence, vol. 206, A.

[5] Abraham, A.-E. Hassanien, A. de Leon, F. de Carvalho, and V. Snasel, Eds, Berlin, Germany: Springer-Verlag, 2009, pp. 371–397.

[6] N. F. Samatova, G. Ostrouchov, A. Geist, and A. V. Melechko, "RACHET: An efficient cover-based merging of clustering hierarchies from distributed datasets," Distrib. Parallel Databases, vol. 11, pp. 157–180, Mar. 2002.

[7] M. Eisenhardt, W. Muller, and A. Henrich, "Classifying documents by distributed P2P clustering," in Proc. Informatik, Sep. 2003, pp. 286 291.

[8] R. Wolff, K. Bhaduri, and H. Kargupta, "A generic local algorithm for mining data streams in large distributed systems," IEEE Trans. Knowl. Data Eng., vol. 21, no. 4, pp. 465–478, Apr. 2009.

[9] K. Tasoulis and M. N. Vrahatis, "Unsupervised distributed clustering," in Proc. IASTED Int. Conf. Parallel Distrib. Comput. Netw., 2004, pp. 347–351.

[10] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, "Approximated clustering of distributed high-dimensional data," in Proc. 9th Pacific-Asia Conf. Adv. Knowl. Discovery Data Min., 2005, pp. 432–441.

[11] L. M. Aouad, N.-A. Le-Khac, and T. M. Kechadi, "Lightweight clustering technique for distributed data mining applications," in Proc. 7th Int. Conf. Data Mining, 2007, pp. 120–134.

[12] S. Merugu and J. Ghosh, "A privacy-sensitive approach to distributed clustering," Pattern Recognit. Lett., vol. 26, no. 4, pp. 399–410, 2005.

[13] A. Elgohary, "Scalable embeddings for kernel clustering on mapreduce," M.Sc. Thesis, Electrical and Computer Engineering, Univ. Waterloo, Waterloo, ON, Canada, 2014.

[14] Eyal, I. Keidar, and R. Rom, "Distributed data clustering in sensor networks," Distrib. Comput., vol. 24, no. 5, pp. 207–222, 2011.

[15] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino, "Fault tolerant decentralised k-means clustering for asynchronous large-scale networks," J. Parallel Distrib. Comput., vol. 73, no. 3, pp. 317–329, 2013.

[16] P. Shen and C. Li, "Distributed information theoretic clustering," IEEE Trans. Signal Process., vol. 62, no. 13, pp. 3442–3453, Jul. 1, 2014