JMLR: Workshop and Conference Proceedings 41:19–32, 2015 BIGMINE 2015

Anytime Concurrent Clustering of Multiple Streams with an Indexing Tree

Zhinoos Razavi Hesabi ZHINOOS.RAZAVI@RMIT.EDU.AU
Timos Sellis TIMOS.SELLIS@RMIT.EDU.AU
Xiuzhen Zhang XIUZHEN.ZHANG@RMIT.EDU.AU
School of Computer Science and IT, RMIT University, Melbourne, Australia

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

With the advancement of data generation technologies such as sensor networks, multiple data streams are continuously generated. Clustering multiple data streams is challenging as the requirement of clustering at anytime becomes more critical. We aim to cluster multiple data streams concurrently and in this paper we report our work in progress. ClusTree is an anytime clustering algorithm for a single stream. It uses a hierarchical tree structure to index micro-clusters, which are summary statistics for streaming data objects. We design a dynamic, concurrent indexing tree structure that extends the ClusTree structure to achieve more granular micro clusters (summaries) of multiple streams at any time. We devised algorithms to search, expand and update the hierarchical tree structure of storing micro clusters concurrently, along with an algorithm for anytime concurrent clustering of multiple streams. As this is work in progress, we plan to test our proposed algorithms, on sensor data sets, and evaluate the space and time complexity of creating and accessing micro-clusters. We will also evaluate the quality of clustering in terms of number of created clusters and compare our technique with other approaches.

Keywords: Distributed data mining, clustering, stream mining, parallel processing

1. Introduction

Advanced technologies, such as sensor networks, social networks, medical applications and so on produce data streams. Data streams are continuously arriving and these can be translated to huge storage with limited processing time. It means that as data is produced, it needs to be mined immediately in a single pass, to be able to answer client queries with minimum response times [1]. In addition, memory for storing arriving data is limited; hence data should be represented as a summary [2]. Even ignoring memory constraints, accessing and maintaining compact data is still a challenge. Having an appropriate data structure is a precondition to be able to accelerate the process of mining streaming data due to memory/disk limitation. For example, Figure 1 illustrates that maintaining data summaries in a tree data structure will minimize average and worst case time required for mining operations (e.g. searching for a proper cluster to insert a new data object). Moreover, data structures that can support dynamic updates as data arrives (insertions and deletions, e.g. insertion of arriving data objects from stream into the data structure)are preferred [3].

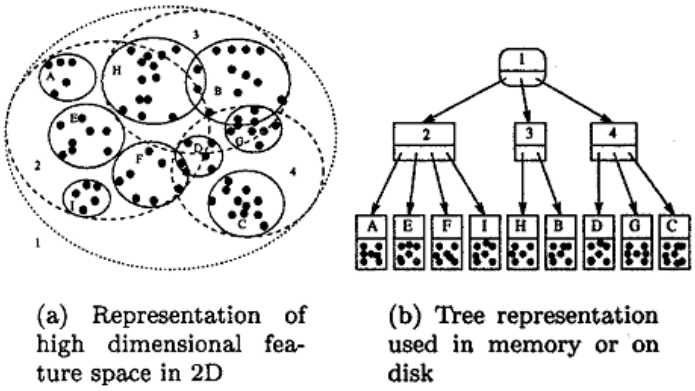

Figure 1: The R-tree Family structure (Source: [3])

With advancement in data collection and generation technologies, such as sensor networks, we are now facing environments that are equipped with distributed computing nodes that generate multiple streams rather than a single stream. Mining a single stream data is challenging therefore mining multiple data streams becomes even more challenging. Some studies have focused on clustering of multiple streams in a centralized fashion, while others have focused on clustering multiple streams in a distributed model [4], [5]. In [6], distributed data mining algorithms, systems and applications are briefly reviewed. Although many parallel and distributed clustering algorithms have been introduced for knowledge discovery in very large data bases [7], [8], scalability of data stream mining algorithms has reached its limitations therefore development of more parallel(concurrent) and distributed mining algorithms is needed. To the best of our knowledge there is not yet any algorithm for clustering multiple streams concurrently to speed up the process of clustering. Therefore, we propose a new framework to cluster multiple streams through a concurrent index data structure from the R-tree family [9].

We have extended the ClusTree algorithm [10] by replacing its data structure and access method from single access to multiple access and removing the buffer used as the anytime clustering feature. More specifically, this paper reports on our project's contribution to multiple data stream clustering. Firstly, we substituted single access method with a multiple (concurrent) access method within the maintenance index data structure. Although there are some constraints on concurrent access to the index data structure, we believe that the concurrent clustering speeds up the process of clustering multiple streams, and creates more clusters at any given time. Secondly, we reduced the space complexity of the ClusTree algorithm by removing its buffers at each entry. Finally, we introduced a new framework to cluster multiple streams (distributed streams) which can also be used for very fast single streaming data.

Through our improvements, we insert data objects to the proper micro-clusters in near real time, through concurrent access. Indeed, the anytime property of the ClusTree allows for interruption of the insertion process when a new data object arrives. The buffered data object should wait till a new data object arrives, then ride down the same subtree where

the data object is buffered. Then, the new and the buffered data objects can descend the tree. Additionally, waiting at the buffer's entry may cause the buffered data object to become obsolete which consequently affects the quality of clustering. We also extract intra-correlation aspects from multiple streams through concurrent access to achieve high quality clusters.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 provides some background on the ClusTree algorithm. Section 4 introduces our proposed multiple stream clustering framework. Section 5 explains our concurrent clustering algorithm in detail. We conclude the paper in Section 6 with a summary of key discussions.

2. Related Work

We do not aim to review all stream clustering methods, we focused on those that are relevant to our research. We divide relevant works on stream clustering into single stream and multiple stream (distributed) groups. We start with a brief introduction to single stream clustering, continued with a few related studies on single stream clustering and finished with reference to a few distributed clustering algorithms.

Stream clustering approaches include two phases : online and offline. The infinite property of data stream and restricted memory make it impossible to store all incoming data. Therefore, summary statistics of data is collected at the online phase, and then a clustering algorithm is performed on obtained summaries at the offline phase. At the online phase, micro-clusters are created to group and store summary information with similar data locality, and these micro-clusters are accessed and maintained through a proper data structure. Micro-clusters are stored away on a disk at a given time for a snapshot of data following a pyramidal time frame to be able to recall summary statistics from various time horizons. This provides further insights into the data through offline clustering.

BIRCH [11] is the pioneering work introducing Cluster Feature(CF) as a compact representation of data. BIRCH is for clustering static data sets (whole data set) rather than evolving data. ClueStream [12] introduced a micro-clustering technique and added some additional information to the BIRCH algorithm in order to adapt with continuous arriving data. This additional information is about timestamps of arriving data streams. DenStream [13] was proposed on a density based two-phase stream clustering algorithm. ClusTree [10] has been developed as the first, anytime clustering algorithm. The main characteristic of ClusTree over other existing micro-clustering algorithms is its adaptability with the speed of streams in an online model. We extended the idea of ClusTree making it applicable to multiple streams. Many of these two-phase clustering algorithms are reviewed in [14] in terms of number of parameters, cluster shape, data structure, window model and outlier detection.

All the aforementioned algorithms are designed and developed to cluster a single data stream. However, with the new generation of distributed data collection and multiple streams acquisition, it is desirable to introduce more parallel and distributed stream mining algorithms to tackle scalability and efficiency limitations of stream mining algorithms. Although many studies have been conducted on distributed clustering algorithms in very large, static data sets [15], [16], few studies have been reported on parallel and distributed stream clustering [5], [17], [18], [4], [19], [20], [21], [22]. None of the above algo-

rithms enable anytime concurrent clustering of multiple streams to speed up the process of clustering or extract intra-inter correlations of multiple streams to achieve high quality clusters.

3. The ClusTree Algorithm

ClusTree [10] is an anytime stream clustering algorithm which groups similar data into the same cluster based on the micro-clustering technique. ClusTree stores N; number of data objects, LS; their linear sum, and SS; their square sum in a cluster feature tuple $CF(N, LS, SS)$ as summary statistics of data. It considers the age of the objects in order to give higher weighting to more recent data. The CF tuples are enough to calculate mean, variance and other required parameters for clustering. Then an index data structure from the R-tree family is created to maintain CFs to speed up the process of accessing, inserting and updating micro-clusters. In this way for an arriving data object, ClusTree descends the tree based on minimum distance between CFs and the arrived data object, to insert the data object into the closest micro-cluster at the leaf level within the given time. If a new data object arrives while the current data object has not yet reached the leaf level to be inserted to a proper micro-cluster within the given time, then its insertion process is interrupted. The interrupted object is left in the buffer of an inner node; the tree is descended to find a path for the new object. The buffered object has a chance to continue descending the hierarchy if it has not been outdated up until a new data object arrives where its path to descend the tree is the same as the buffered object. Therefore, the buffered object descends the tree along with the new object as a hitchhiker to be inserted into the most similar micro-cluster at the leaf level. Using the buffer makes ClusTree able to adapt with the speed of data stream to insert data objects into the micro-clusters at any given time. Moreover, ClusTree deals with high speed data stream by aggregating data objects at the top level of the tree, then inserting aggregated objects into the proper micro-clusters.

Figure 2 [10] shows the inner entry and leaf entry in a ClusTree. Each entry in an inner node stores CF of objects and has a buffer to store CFs of interrupted objects which may be empty. Additionally, each entry keeps a pointer to its child. Entry of each leaf node only stores a CF of the object(s) it represents [10].

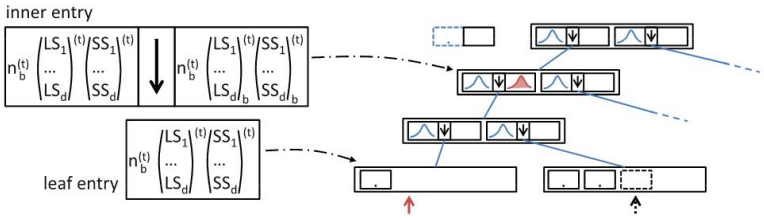

Figure 2: Inner node and leaf node structure in ClusTree (Source: [10])

Figure 3 shows the overall algorithmic scheme of the ClusTree algorithm. The micro-clusters are stored at particular moments in the stream, which are referred to as snapshots. The offline macro-clustering algorithm will use these finer level micro-clusters in order to create higher level clusters over specific time horizons.

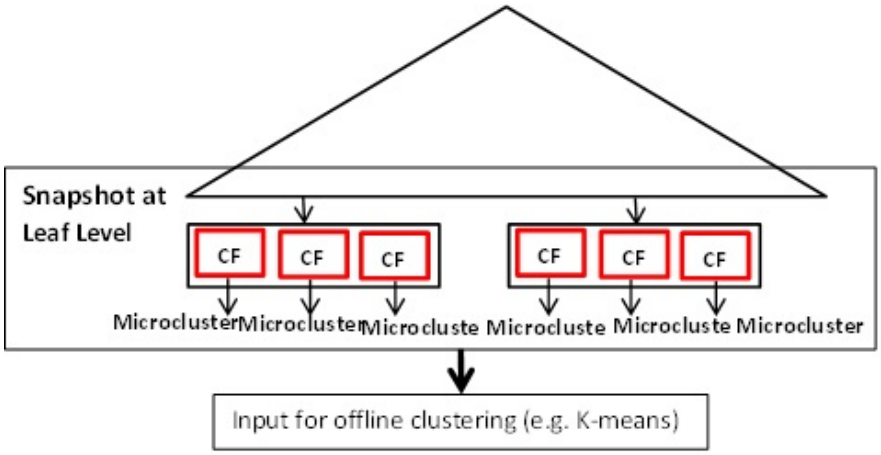

Figure 3: A snapshot of micro-clusters in the ClusTree

The ClusTree algorithm is proposed to cluster a single data stream with varying inter-arrival times. We have proposed a new algorithm based on the ClusTree to cluster multiple data streams concurrently. We explain our proposed algorithm in detail in the next section.

4. Anytime Concurrent Multi-Stream Clustering using an Indexing Tree

Data streams are continuously produced and need to be analysed online. Moreover, multi-stream applications demand higher anytime requirements due to streams arriving at any time and with varying speeds. This continuously arriving data means huge storage requirements. Therefore, online multi-stream clustering is a twofold problem in terms of time and space complexity. For space complexity, many studies have been conducted to represent distribution of data in a compact way. The main idea is that instead of storing all incoming objects, summary statistics of data objects will be stored to reduce storage problem. Many techniques are proposed in the literature to achieve summary of data. One of these techniques is called cluster feature vector which we use in our proposed algorithm to obtain summaries of data objects. The other issue is related to accessing these summary statistics, which is crucial in terms of time complexity. Therefore, choosing a proper data structure plays an important role in maintaining and updating these summary statistics in memory. In fact, these summaries are generated and maintained in a proper data structure in real time, and then are stored away on a disk for further and future analysis called offline processing. Hence, to achieve "extreme" anytime clustering, we propose to extend the ClusTree structure to a concurrent hierarchical tree structure to index more granular micro-clusters.

Figure 4 shows a general view of our proposed framework in which each stream is assigned to one processor. All the processors have equal access to a shared memory. The processors will create micro-clusters in memory in a parallel way through concurrency control. We expect to create more accurate micro-clusters with high granularity, in contrast to serialized clustering of a single stream using the ClusTree. Granularity is considered in terms of number of micro clusters. Intuitively, high accurate clusters will be created by

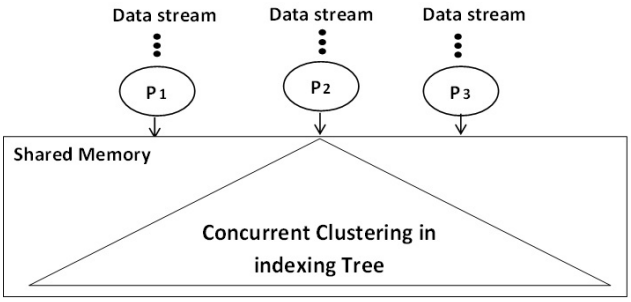

Figure 4: Proposed concurrent clustering framework

extracting correlations among different data streams through concurrent clustering. Similar data objects from different data streams have more chances to be grouped into the same cluster compared with local clustering of individual streams with a decentralized model.

Like many optimization problems using a search tree to an obtain optimal solution, a tree of micro-clusters is created in which clustering data objects is started from the root. The children of the root are obtained by splitting the root into small clusters. The leaves of a tree represent micro-clusters in a given time interval. The goal of this paper is to insert data objects concurrently into their closest micro-clusters; optimal leaves, by using an index search tree. The cost of searching the tree and adding a data object to the closest cluster is $O(log(n))$, where n is the number of elements in the tree. Using a parallel algorithm with concurrency control seams to increase the level of granularity and reduce the execution time of creating micro-clusters. To achieve this, each processor can explore a set of subtrees to reach proper micro-clusters. However, a tree is created during the exploration which means that subtrees are not assigned to each processor in advance. Each processor will get the unexplored nodes from a Global Data Structure(GDS) [23].

We propose to use a search tree that allows concurrent access to the GDS in the context of parallel machine with shared memory in order to create and maintain high granularity micro-clusters. Each processor will process clustering operations on the GDS, stored in the shared memory. The main difficulty is to keep the GDS consistent, and to allow the maximum level of concurrency. In the shared memory model, the GDS is stored in the shared memory which can be accessed by each processor. The higher the access concurrency, the higher the granularity of clustering. The main issue is the contention access to the data structure. Mutual exclusion is performed to provide data consistency. We suggest using a concurrent index structure from the R-tree family to create and maintain more micro clusters with high accuracy from multiple streams.

The idea of creating micro-clusters at the leaf level, means that the algorithm can take a snapshot and send the results to any offline clustering as with the ClusTree algorithm. However, it should be emphasized that ClusTree is applied on a single stream in a serialized model while our proposed algorithm is applied on multiple streams in a parallel model.

Figure 5 compares our proposed concurrent clustering of multiple streams with the ClusTree algorithm.

As described in Section 3, ClusTree uses a buffer for each entry of each node to do anytime clustering. As an example of anytime clustering of ClusTree, suppose that data object 1 arrives at timestamp t. Meanwhile data object 1 is descending the tree to find the proper micro-cluster. Data object 2 arrives at timestamp t+1. The insertion of data object 1 is interrupted in the middle of its path in the tree, for example at level i which is not the leaf level. Data object 1 is added to the buffer's entry of node on level i. Then data object 2 descends the tree. Data object 1 is waiting at the buffer to be picked up by a new arriving data object. Data object 1 can be successfully inserted to an appropriate micro-cluster, provided that data object 1 and the new arriving data object belong to the same subtrees. Otherwise, data object 1 might be obsolete and deleted.

In our proposed concurrent clustering, as can be seen in Fig 5, arriving new data objects do not interrupt the insertion process of the current data object, except when they need to modify the same leaf node. In this situation, the leaf node will be write-locked and just one of the data objects has access to this part of the shared memory. Therefore, intuitively, data objects from multiple streams can descend the tree through different subtrees. In this way, data objects have more opportunity to be added to the closest micro-clusters in near real time.

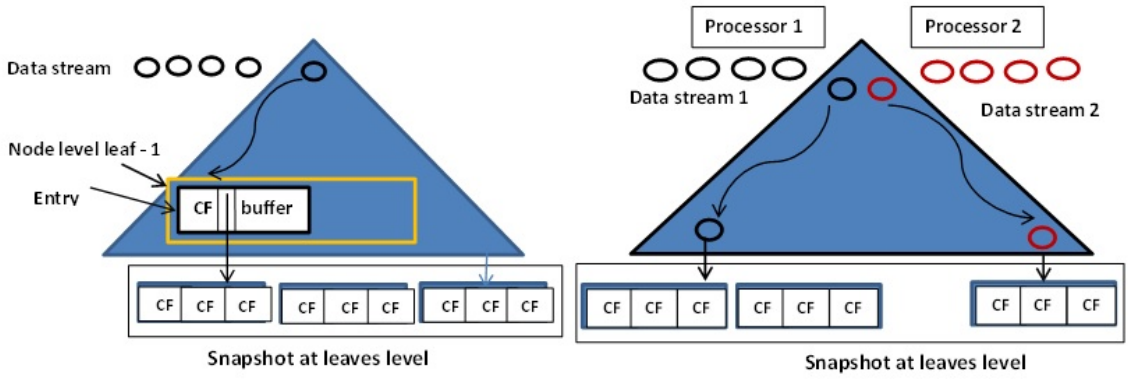

Figure 5: Comparison of ClusTree (Left) and Proposed Concurrent Clustering (Right)

5. The Anytime Concurrent Clustering Algorithm

Our proposed clustering algorithm is based on using micro-clusters to present data distribution in a compact way. Micro-clusters are broadly used in stream clustering to create and maintain a summary of the current clustering. A micro-cluster stores summary statistics of data objects as a cluster feature tuple CF instead of storing all incoming objects. A cluster feature tuple (N, LS, SS), has three components. N is the number of represented objects, LS is the linear sum and SS is the squared sum of data objects. Maintaining these summary statistics is enough to calculate mean, variance and other parameters such as centroid and radius, as follows.

$$Centroid:\ \overrightarrow{x_0} = \frac{\sum_{i=1}^{N} \overrightarrow{x_i}}{N}$$

$$Radius:\ R = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\overrightarrow{x_i} - \overrightarrow{x_j})}{N} \right)^{\frac{1}{2}}$$

Each cluster feature represents a micro-cluster of similar data objects with the following properties.

Additivity Property of *CF*: CF has the property of additivity which means if $\mathrm{CF}_1 = (\mathrm{N}_1, LS_1, SS_1)$ and $\mathrm{CF}_2 = (\mathrm{N}_2, LS_2, SS_2)$ then $\mathrm{CF}=\mathrm{CF}_1+CF_2 = (\mathrm{N}_1+N_2, LS_1+LS_2, SS_1+SS_2)$.

Subtractive property of *CF*: CF has the subtractive property which means that if $\mathrm{CF}=(N, LS, SS)$ and $\mathrm{CF}_1 = (\mathrm{N}_1, LS_1, SS_1)$ then CF -$\mathrm{CF}_1 = (\mathrm{N}-\mathrm{N}_1, LS\text{-}LS_1, SS\text{-}SS_1)$.

These properties of cluster features are used when a cluster feature tuple requires an update. As an example, when two micro-clusters are merged, the cluster feature of the merger is calculated using additivity property.

We extend the ClusTree algorithm into a parallel model in order to cluster multiple streams concurrently. We propose the use of a concurrent index structure from the R-tree family to maintain cluster features in a hierarchical structure. As in all such tree structures, internal nodes hold a set of entries between m and M (fanout) while the leaf nodes similarly store a number of entries between l and L. Figure 6 shows the details of internal node's entries and leaf node's entries of our proposed tree structure. An entry of an internal node contains [*CF* (N, LS, SS), Child-ptr, *LSN*], where *CF* is a cluster feature of data object(s), Child-ptr is a pointer to its child node and *LSN* is a logical sequence number. *CF* is calculated for each dimension of the data object. For a d-dimensional data object, the linear square and sum square are calculated for all d-dimensions. An entry in a leaf contains a cluster feature of data object(s), and *LSN*.

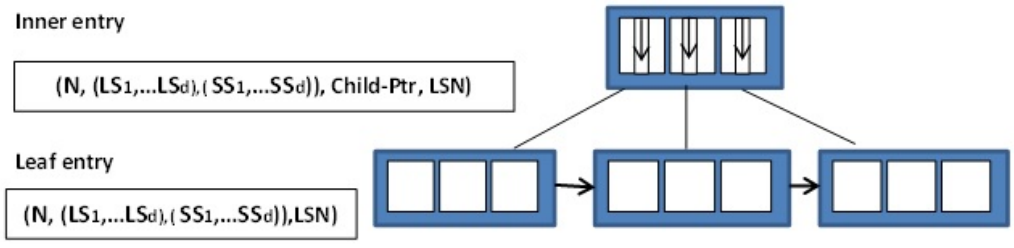

Figure 6: Internal node and leaf node structure of proposed concurrent clustering

The hierarchy of our concurrent clustering scheme is created like an R-tree except that cluster features are stored instead of bounding rectangles. Incoming data objects are clustered accordingly. First, we have to find the proper micro-cluster to insert an arriving data object into. To achieve this, a data object descends the tree by starting from the root. At each node, the distance between CF of the data object and *CF* of the node's entries are calculated. The entry with the closest distance is selected. The selected entry has a pointer to its child, so the data object descends the tree using the pointer. The data object descends the tree towards the leaf level for a proper micro-cluster. When descending the tree, the timestamp of the visiting node is updated.

As illustrated in Figure 6, both the node and its entries have certain capacity. This means that before a data object is inserted to the closest entry at the leaf level, capacity of the closest entry is checked. Different scenarios occur. First, the closest entry(proper micro-cluster) has enough space for the data object. After an insertion, the cluster feature of the entry will be updated through the additivity property of *CF*. Second, the closest entry does not have enough space to insert the data object. In this situation, the capacity of the node containing the closest entry is checked. If the node has enough space, a new entry is created to insert the data object into. Then, a new entry at the parent of the node should be created to point to the created new entry at the node. Finally, if neither the closest entry nor its node have enough space for inserting a data object, the node will be split. Splitting a node means a new node is created which needs a parent to point to it. This splitting to create parent entry could be continued at upper levels of the tree till the root. If the root is split, then the height of the tree will be increased by one.

In our concurrent clustering, in order to recognize node splitting, we use right-link approach similar to the concurrent R-tree [9]. Suppose that the data object 1 from data stream 1 and data object 2 from data stream 2 are concurrently descending the tree to be inserted into their closest micro-clusters. Data object 1 reaches leaf level and is inserted into a closest entry of leaf node, but this insertion causes a split. Another data object 2 reaches the same leaf node and wants to be inserted into the split node. If the leaf node has been split and data object 2 does not recognize this split, and to be able to traverse this dynamic tree correctly, likewise R-link-Tree, we modify the ClusTree into the concurrent version by adding extra features.

First, Logical Sequence Number(LSN)(as shown in Fig 6) is assigned to each node to recognize the split. Second, we link all nodes at each level of the tree using a link list. Using LSN allows the split to be recognized and helps to decide how to traverse the tree. Also linking all nodes at each level of a tree enables movement to the right of a split node.

Figure 7 presents an example where a node is split and the right-link along with LSN is used to chain the split. One of the properties of the R-link-tree data structure is order insensitivity. As can be seen in Fig 7, it is possible that node P_1 is ordered before node P_2 (from left to right at each level) but because of a split, the child of P_1, C_4, is visited after child of P_2, C_1.

Using a global counter, each node has its unique LSN. Every entry of each node and its child's entries have the same LSN. In the occurrence of a split, a new right sibling node

will be created for the split node. The LSN of a split node is given to the new right sibling and a new LSN is assigned to the split node. A data object descending the tree recognizes the split by comparing LSN of a visiting(parent) node and its child node. If LSN of the parent and its child is equal, no split has occurred; otherwise if LSN of the child node is greater than its parent node, it means there is a split and the clustering process moves right of the child node till it visits a node with the same LSN of the parent node, showing the furthest right node split off the old node. The possibility of moving right to the split node is provided by using a link-list of nodes at each level of the hierarchy.

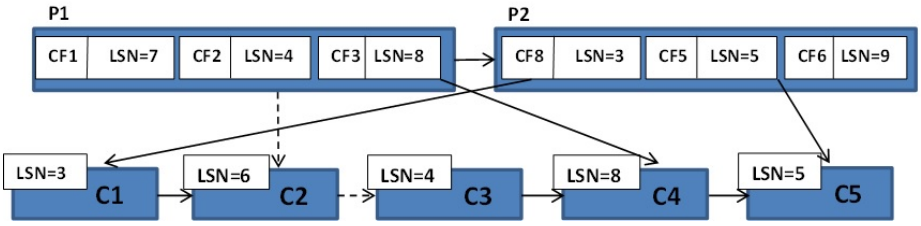

Figure 7: Node split recognition using LSN and right-link

For concurrency control, we use a lock-coupling technique in such a way that during the process of traversing the tree, nodes are read-locked. Hence, data objects from different streams can access the tree and descend the tree in parallel. The main issue is at the time of inserting a data object into a micro-cluster, then updating the CF of its parent at the upper level of the tree. To solve this problem like in a R-link-tree, we use the write-lock; when a data object is being inserted into a leaf node, the leaf node is locked. After an insertion, the CF of the node's parent should be updated. Therefore, the parent is locked and the leaf node is unlocked.

Algorithm 1: *Clustering Algorithm*

Input: D-dimensional data objects $O_i, O_j, ...$
Output: Inserting data objects $O_i, O_j, ...$ into the closest micro-clusters

```
for all processors P_i, P_j, ... do
    ClosestMicrocluster = searchLeaf(root, O, root-lsn)
    insert O on ClosestMicroCluster at leaf
    if leaf was split then
        expandParent(leaf,CF(leaf),LSN(leaf),right-sibling,CF(right-sibling),
        LSN(right-sibling));
    else
        if CF of leaf changed then
            updateParent(leaf, CF(leaf));
        else
            w-unlock(leaf);
        end
    end
end
```

Our main proposed concurrent clustering algorithm (as shown in Algorithm 1) consists of a search process to find the closest micro-cluster, updating the *CF* of the parents after clustering, expanding the parents in the case of split child and installing a new entry for this split at upper levels of the tree. The algorithm is the same as the concurrent R-tree algorithm [9] except that our purpose is to manage the micro-clusters. We explain each function in details as follows.

searchLeaf: The searchLeaf function is called at the beginning of the clustering algorithm (1) to find the closest micro-cluster for a given data object at the leaf level. The searchLeaf function starts the process of a search from the root. During the process of searching the tree, if a visiting node is not leaf, it is read-locked. Otherwise, it is write-locked. For each node, the *LSN* of the visiting node is compared with the *LSN* of its parent. If the *LSN* of the parent is smaller than the *LSN* of the visiting node, a split has occurred. Therefore, the tree is traversed to the right of the visiting node(split node) till finding a node with the *LSN* equals to the *LSN* of the parent guarantee to find the closest entry even after a split. If the split node is at the leaf level, then the searchLeaf function returns the closest entry as the closest micro-cluster to the clustering algorithm. Otherwise, the process of search keeps descending the tree recursively from the child of the closest entry and the visited node is read-unlocked.

expandParent: After finding the closest micro-cluster through serachLeaf function, the data object is inserted into. If the insertion of the data object causes a split, then the expandParent function is called. The expandParent function either installs a new entry as the parent of the new created leaf (because of the split) at the top level of the split leaf or find an entry for the new created leaf in the parent of the split leaf node or its right sibling. The former is a new split at the parent level of the split node. Therefore, the expandParent function is recursively called up until the root is split or no more split is happened. During the process of expanding a parent, the child nod is write-locked till the parent is accesses. Then, the child node is write-unlocked and the parent is write-locked.

updateParent: Whenever a data object is inserted into the leaf node and its CFs is updated, or CFs of a parent is updated because of a split, the updateParent function is called to propagate these updates up to the parent's levels.

We aim to optimize the process of clustering by finding top-k closest micro-clusters. This means that descending the tree by finding the closest entry among all entries of visiting nodes does not guarantee arrival at the closest micro-cluster among all other micro-clusters at leaf level. Therefore, to find global optimum; the closest micro-cluster among all micro-clusters, we use a stack data structure to keep track of the top-k closest entries to a data object. In order to maintain up-to-date clustering, we use a buffer in each node, whenever a new data object arrives and descends the tree, the time stamp of the visiting node is updated like ClusTree.

6. Discussion

In this work in progress, we proposed a new, anytime, concurrent, multiple stream clustering algorithm using an indexing tree. Our proposed algorithm is based on one of the well-known micro-clustering technique, ClusTree [10]. We captured the summary statistics of multiple data streams concurrently in the online phase. We proposed to maintain statistical information of the data locality in micro-clusters at a dynamic, multiple access index data structure for further offline clustering. In the online phase, the index data structure maintains summaries of data in the format of cluster feature tuples (CF) instead of storing all incoming objects. Then, the data structure is traversed through an index to insert new data objects concurrently into their closest micro-clusters. We designed the concurrent clustering algorithm and will further develop this on SAMOA [24]. To evaluate our algorithm, we plan to test our proposed algorithm on two real data sets: 1) the environmental sensor data set with 97 stations and 18 attributes, which is available from [25], and 2) the Forest Covertype dataset [26]. We will assess our proposed clustering algorithm and compare it with competing clustering algorithms, including ClusTree. Our experimental analysis will include time and space complexity of creating and maintaining concurrent clustering trees in terms of the number of generated micro-clusters and quality of clustering. We plan to experiment with three different workloads.

1) High workload which consists of receiving two data streams with high speed
2) Moderate workload of receiving a slow speed stream and a high speed stream
3) Low workload receiving two slow speed data streams

We also plan to study the effect of the number of processors required to perform efficient, concurrent clustering.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, (New York, NY, USA), pp. 1–16, ACM, 2002.

[2] C. Aggarwal and P. Yu, "A survey of synopsis construction in data streams," in *Data Streams* (C. Aggarwal, ed.), vol. 31 of *Advances in Database Systems*, pp. 169–207, Springer US, 2007.

[3] D. White and R. Jain, "Similarity indexing with the ss-tree," in *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, pp. 516–523, Feb 1996.

[4] A. Guerrieri and A. Montresor, "Ds-means: Distributed data stream clustering," in *Euro-Par 2012 Parallel Processing* (C. Kaklamanis, T. Papatheodorou, and P. Spirakis, eds.), vol. 7484 of *Lecture Notes in Computer Science*, pp. 260–271, Springer Berlin Heidelberg, 2012.

[5] J. a. Gama, P. P. Rodrigues, and L. Lopes, "Clustering distributed sensor data streams using local processing and reduced communication," *Intell. Data Anal.*, vol. 15, pp. 3–28, Jan. 2011.

[6] S. Parthasarathy, A. Ghoting, and M. Otey, "A survey of distributed mining of data streams," in *Data Streams* (C. Aggarwal, ed.), vol. 31 of *Advances in Database Systems*, pp. 289–307, Springer US, 2007.

[7] X. Xu, J. Jger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 263–290, 1999.

[8] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313 – 1325, 1995.

[9] M. Kornacker and D. Banks, "High-concurrency locking in r-trees," in *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, (San Francisco, CA, USA), pp. 134–145, Morgan Kaufmann Publishers Inc., 1995.

[10] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The clustree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, 2011.

[11] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, (New York, NY, USA), pp. 103–114, ACM, 1996.

[12] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pp. 81–92, VLDB Endowment, 2003.

[13] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*, pp. 328–339, 2006.

[14] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Comput. Surv.*, vol. 46, pp. 13:1–13:31, July 2013.

[15] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313 – 1325, 1995.

[16] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Min. Knowl. Discov.*, vol. 3, pp. 263–290, Sept. 1999.

[17] A. Zhou, F. Cao, Y. Yan, C. Sha, and X. He, "Distributed data stream clustering: A fast em-based approach," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 736–745, April 2007.

[18] G. Cormode, S. Muthukrishnan, and W. Zhuang, "Conquering the divide: Continuous clustering of distributed data streams," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 1036–1045, April 2007.

[19] P. P. Rodrigues and J. Gama, "Distributed clustering of ubiquitous data streams," *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, vol. 4, no. 1, pp. 38–54, 2014.

[20] A. T. Vu, G. D. F. Morales, J. Gama, and A. Bifet, "Distributed adaptive model rules for mining big data streams," in *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pp. 345–353, 2014.

[21] M.-Y. Yeh, B.-R. Dai, and M.-S. Chen, "Clustering over multiple evolving streams by events and correlations," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, pp. 1349–1362, Oct 2007.

[22] B.-R. Dai, J.-W. Huang, M.-Y. Yeh, and M.-S. Chen, "Adaptive clustering for multiple evolving streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, pp. 1166–1180, Sept 2006.

[23] B. Cun and C. Roucairol, "Concurrent data structures for tree search algorithms," in *Parallel Algorithms for Irregular Problems: State of the Art* (A. Ferreira and J. Rolim, eds.), pp. 135–155, Springer US, 1995.

[24] G. D. F. Morales and A. Bifet, "Samoa: Scalable advanced massive online analysis," *Journal of Machine Learning Research*, vol. 16, pp. 149–153, 2015.

[25] E. D. Sensors, "Environmental data: Sensors." http://lcav.epfl.ch/page-86035-en.html. [Online; accessed 20-03-2015].

[26] H. S and B. S, "The UCI KDD archive.." https://archive.ics.uci.edu/ml/datasets/Covertype, 1999. [Online; accessed 20-03-2015].