Chapter 5

Distributed Grid-Density based Clustering

Identifying clusters in large spatial data is a difficult task due to the high amount of processing time needed in handling voluminous data. Distributed and parallel clustering approaches help to reduce the time needed by distributing the processing to different machines. This also improves the response time.

In this chapter, we discuss two distributed clustering techniques that can be applied to handle large scale 2D spatial datasets and large satellite image datasets with high resolution. The first technique (DGDCT) aims to identify embedded clusters in any large 2D spatial datasets with improved clustering quality. The second method is a distributed Grid-Density based Satellite data Clustering technique, DisClus, that can detect clusters of arbitrary shapes and sizes over large, high resolution, multi-spectral satellite datasets. Both techniques are implemented using a client server approach, where the huge dataset stored in the server is partitioned into almost $k_p$ equal partitions that are used by $k_p$ clients to identify the clusters in parallel for each partition. Finally, the clusters obtained from the $k_p$ clients are merged at the server for the final results. Experimental results establish the superiority of the techniques in terms of scale-up, speedup as well as cluster quality, in comparison to similar algorithms.
5.1 Introduction

Extraction of hidden information from huge datasets is a challenging task in data mining. With the increase in the amount of spatial data, the need for efficient and effective spatial data mining techniques is of utmost importance. Though traditional data mining algorithms may be applicable in some spatial datasets, the challenges imposed by the huge amount of spatial data, need to be addressed. The huge size of datasets, its wide distribution over several sites and the computational complexity are the factors contributing towards the development of parallel and distributed algorithms in the data mining domain. Clustering is the process of division of a dataset into subsets or clusters, so that the similarity of points in each partition is as high as possible while points in different partitions are dissimilar. Parallel and distributed spatial data clustering algorithms may help in addressing the problem mentioned before. Distributed clustering is the partitioning of data into groups, in a distributed environment. Although this field is relatively new, yet it has been explored intensively in the last few years, as the need to employ distributed algorithms has grown significantly. The evolution of the networking and storage equipment fostered the development of very large datasets and it is infeasible to centrally process these datasets in order to analyze them. Distributed clustering is applied when either the data that need to be processed is distributed, or the computation is distributed, or both of them. If none of these two is distributed, then it is centralized clustering. Parallel and distributed computing is expected to relieve current clustering methods from the sequential bottleneck, provide the ability to scale massive datasets and improve the response time. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged.

Although it is common for data to be distributed in a parallel/distributed environment, the distribution is governed solely by performance considerations. Three main architectures can be proposed for building parallel/distributed DBMSs.

1. In a shared-memory system, multiple CPUs are attached to an interconnection network and can access a common region of main memory. A shared memory

\footnote{http://www.cs.ucf.edu/courses/cop4710/spr2006/notes.html}
system is illustrated in Figure 5.1.

2. In a shared-disk system, each CPU has a private memory and direct access to all disks through an interconnection network. Figure 5.2, shows a shared-disk system.

3. In a shared-nothing system as shown in Figure 5.3, each CPU has local main memory and disk space, but no two CPUs can access the same storage area; all communications between CPUs are through a network connection.

Shared memory architecture\textsuperscript{2} usually has a block of random access memory that can be accessed by different central processing units (CPUs) in a multiple-processor computer system. This type of architecture is quite easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to the same location. The problem with these systems is that many CPUs need fast access to memory and will likely cache memory, which has two complications: (i) CPU-to-memory connection becomes a bottleneck and shared memory computers cannot scale very well. (ii) Whenever one cache is updated with information by a particular processor, the change needs to be reflected to the other

\textsuperscript{2}http://en.wikipedia.org/wiki/Shared_memory
Figure 5.2: The Shared-disk architecture

Figure 5.3: The Shared-nothing architecture
processors, otherwise the different processors will be working with incoherent data. If these issues are handled the system works well, provide extremely high-performance access to shared information between multiple processors. On the other hand they can sometimes become overloaded and become a bottleneck to performance.

In shared-disk architecture\textsuperscript{3} all processors can access the same disks with about the same performance, but are unable to access each other’s RAM. With the advent of Network Attached Storage devices (NAS) which allow a storage device on a network, is to be mounted by a set of nodes, shared disk has become increasingly popular. One key advantage of shared-disk systems over shared-nothing is in usability, since DBAs of shared-disk systems do not have to consider partitioning tables across machines. In this system the failure of a single DBMS processing node does not affect the other nodes’ ability to access the full database which is not the case with shared-memory systems (that fail as a unit), and shared-nothing systems (that lose at least some data upon a node failure).

Shared nothing architecture (SNA)\textsuperscript{4} is a distributed computing architecture consisting of multiple nodes such that each node has its own private memory, disks and input/output devices independent of any other node in the network. Each node is independent and self-sufficient, and shares nothing across the network. Therefore, there are no points of contention across the system and no scope for data sharing or system resources. This type of architecture is highly scalable and has become quite popular for web development because of its scalability. An SN system typically partitions its data among the different nodes such that each node may be responsible for handling a particular task (interacting with different types of users, handling different types of queries, serving different geographic areas etc), or it may require every node to maintain its own copy of the application’s data, using some kind of coordination protocol to interact with other nodes as required.

The disadvantage of the shared-memory and shared-disk architectures is interference. As the number of CPUs are increased, existing CPUs are slowed down because

\textsuperscript{3}http://www.coders2020.com/explain-the-shared-disk-architecture
\textsuperscript{4}http://en.wikipedia.org/wiki/Shared_nothing_architecture
of the increased contention for memory accesses and network bandwidth. The shared-nothing architecture requires more extensive reorganization of the DBMS code, but it has been shown to provide a linear speed-up and linear scale-up. Linear speed-up occurs when the time required by an operation decreases in proportion to the increase in the number of CPUs and disks. Linear scale-up occurs when the performance level is sustained if the number of CPUs and disks are increased in proportion to the amount of data. As a result, ever-more-powerful parallel database systems can be constructed by taking advantage of the rapidly improving performance for single-CPU systems and connecting as many CPUs as desired.

In this chapter, we propose two distributed clustering techniques that use the shared-nothing architecture. The first technique presented in Section 5.3 is capable of identifying arbitrary shaped embedded clusters as well as multi-density clusters over large spatial datasets. The second technique is reported in Section 5.4 and has been applied over huge satellite images to detect the various clusters present in the data.

5.2 Related Work

This section presents a selected survey on some of the distributed and parallel algorithms.

5.2.1 Distributed and Parallel Clustering Techniques

For the past few decades the mainstream data clustering technologies have been fundamentally based on centralized operation; datasets were of small manageable sizes, and usually resided on one site that belonged to one organization. Today, data is of enormous sizes and is usually located on distributed sites; the primary example being the Web. This created a need for performing clustering in distributed environments. Distributed clustering solves two problems: infeasibility of collecting data at a central site, due to either technical and/or privacy limitations, and intractability of traditional clustering algorithms on huge datasets.
In [BBD04], a parallel implementation of the DBSCAN algorithm based on low cost distributed memory multi-computers is presented. Here, a centrally located dataset is spatially divided into nearly equal partitions with minimum overlap. Each such partition is sent to one of the processors for parallel clustering. The clustering results of the partitions are then collected by the central processor in an orderly manner and they are merged together to obtain the final clustering. The algorithm is scalable both in terms of speedup and scale-up and significantly reduces the computation time.

In [DM99], a parallel version of the k-means algorithm was proposed based on shared nothing architecture. This algorithm was designed based on the Single Program Multiple Data (SPMD) model having several processors, each having its own local memory, connected together with a communication network. Each processor, or node, receives only a segment of the data that needs to be clustered. One of the nodes selects the initial cluster centroids, before sending them to the others. New distances between centroids and data points are computed independently, but after each iteration of the algorithm, the independent results must be aggregated or reduced. This is done using the MPI (Message Passing Interface). The reduced centroids obtained after the last iteration represent the final result of the clustering process.

Another parallel version of DBSCAN, called PDBSCAN [XJK99], also uses a shared-nothing architecture with multiple computers interconnected through a network. Here, as a data structure, the dR*-tree was introduced which is a distributed spatial index structure in which the data is spread among multiple computers and the indexes of the data are replicated on every computer. The master distributes the entire dataset to every slave. Each slave locally clusters the replicated data and the interference between computers is minimized due to local access of data. The slave-to-slave and master-to-slaves communication is done via message passing. The master manages the task of dynamic load balancing and merges the result produced by the slaves. PDBSCAN offers nearly linear speedup and has excellent scale-up and size-up behavior.
In [JKP03], a Density Based Distributed Clustering (DBDC) algorithm was presented which can be used in the case when the data to be clustered is distributed and infeasible to centralize. DBDC works by first clustering the data locally at different sites independent of each other. The aggregated information about locally created clusters are extracted and transmitted to a central site. On the central site, a global clustering is performed based on the local representatives and the result sent back to the local sites. The local sites update their clustering based on the global model, that is, merge two local clusters to one or assign local noise to global clusters. For both the local and global clustering, density-based algorithms are used. This approach is scalable to large datasets and gives clusters of good quality.

In [FLPT00], a parallel version of the AutoClass system, P-AutoClass is described. In [JK99], a Collective Hierarchical Clustering (CHC) algorithm is reported for analyzing data that is heterogeneously distributed, with each site having only a subset of all features. First, a local hierarchical clustering is performed on each site. Afterwards, the obtained dendrograms are sent to a facilitator which computes the global model, using statistical bounds. The aggregated results are similar to centralized clustering results, making CHC an exact algorithm.

The algorithm P2P K-means, developed by Datta et. al. [DGK06] is one of the first algorithms developed for P2P systems. Each node requires synchronization only with the nodes that it is directly connected to, or its neighborhood. Only one node initializes the centroids used for k-means, which are then spread to the entire network. The centroids are updated iteratively. Before computing them at step i, a node must receive the centroids obtained at step i - 1 by all of its neighbors. When the new centroids of a particular node do not suffer major modifications, then the node enters a terminated state, where it doesn't request any centroids, but it can response to requests by neighbors. Node or edge failures and additions are also accounted for by P2P k-means, making it suitable for dynamic networks. P2P K-Means algorithm was proposed in [BGM+06] for distributed clustering of data streams in a peer-to-peer sensor network environment.
Jin R. et al. [JGA06] presented a distributed version of Fast and Exact K-Means (FEKM) algorithm, which collected sample data from each data source, and communicated it to the central node. The main data structure of FEKM i.e. the cluster abstract table is computed and sent to all data sources to get global clusters.

In [ALKK07], the authors proposed a lightweight distributed clustering technique based on a merging of independent local sub clusters according to an increasing variance constraint. The key idea of this algorithm is to choose a relatively high number of clusters locally, or an optimal local number using an approximation technique, and to merge them at the global level according to an increasing variance criterion which requires a very limited communication overhead.

Le-Khac N. et al. [LKAK07] presented an approach for distributed density-based clustering. The local models are created by DBSCAN at each node of the system and these local models are aggregated by using tree based topologies to construct global models.

In [TMEDF08], the authors introduced a method to define intuitionistic fuzzy partitions from the result of different fuzzy clustering algorithms such as FCM, entropy based FCM and FCM with tolerance. In this approach, the intuitionistic fuzzy partition permits to cope with the uncertainty present in the execution of different fuzzy clustering algorithms with the same data and with the same parameterization.

In [DGK09], Datta et. al. propose two approximate K-means clustering algorithms that work on uniformly sampled peers. The first algorithm is designed to operate in a dynamic P2P network that can produce clusterings by local synchronization only. The algorithm has been observed empirically to produce accurate clustering results with respect to centralized K-means clustering. However, it cannot offer an analytical accuracy guarantee. Therefore, the second algorithm is proposed which works by taking a uniform random sample of nodes from a static P2P network. This algorithm provides an analytical accuracy guarantee.
Another intuitionistic fuzzy based distributed clustering algorithm is presented in [VTPlO] for homogeneously distributed datasets. The process is carried out in two different levels: local level and global level. In local level, numerical datasets are converted into intuitionistic fuzzy data. Modified fuzzy C-Means algorithm is then used to cluster this data independently from each other. In global level, global centroid is computed by clustering all local cluster centroids. The global centroid is again transmitted to local sites to update the local cluster model.

5.2.2 Discussion

Based on our selected survey and experimental analysis, it has been observed that density based approach is most suitable for quality cluster detection over massive datasets. Almost all clustering algorithms require input parameters, determination of which are very difficult, especially for real world datasets containing high dimensional objects. Moreover, the algorithms are highly sensitive to those parameters. Distribution of most of the real-life datasets are skewed in nature, so, handling of such datasets for all types for qualitative cluster detection based on a global input parameter seems to be impractical. Also handling high dimensional data is a challenging task. The performance of most of the algorithms aimed to identify quality clusters for 2D spatial data degrades with the increase in dimensionality. Algorithms like DBSCAN [EKSX96] and GDBSCAN [SEKX98], which give good quality clusterings, do not work for high dimensional data. Often, the algorithms present in the literature can be found to identify clusters over large spatial data at an abstract level, however, some applications demand for identification of these at a more detailed or finer level. None of the techniques discussed above, is capable to handle the embedded or intrinsic cluster detection problem over massive datasets successfully.

An algorithm which is capable of handling voluminous data and at the same time effectively detects nested or embedded clusters in presence of noise is of utmost importance. The grid density based clustering algorithm (GDCT) discussed in Chapter 3 finds clusters according to the structure of the embedding space. For handling massive datasets, a distributed clustering technique based on GDCT is presented which can effectively address the scalability problem. Better speedup and scale-up are the major attractions of the proposed technique.
5.3 Distributed Grid-Density based Clustering Technique (DGDCT)

This section presents a Distributed Grid-Density based Clustering Technique (DGDCT) capable of identifying arbitrary shaped embedded clusters as well as multi-density clusters over large spatial datasets. For handling massive datasets, we implemented our method using a shared-nothing architecture where multiple computers are interconnected over a network. We consider a system having \( k_p \)-nodes where the entire dataset \( D \) is located in any of the nodes (say initiator node). DGDCT can be initiated in any of the available nodes (computers). The initiator node starts a partitioning strategy thereby dividing the whole dataset into partitions and then distributing the partitions to each of the available computers on the network (one partition is also retained by itself). The initiator node executes a fast partitioning technique to generate the \( k_p \) initial partitions. The partitions are then sent to \( k_p \) nodes (including itself) for cluster detection using a grid-density based clustering technique (GDCT) which can operate over variable density space. Every node clusters only its local data. The initiator node manages the task of dynamic load balancing. Finally, the local cluster results are received from the nodes at the initiator node and a merger module is invoked to obtain the final cluster results. Basically the technique works in three phases and the output of the previous Phase becomes the input of the current Phase. Next, we describe the architecture as shown in Figure 5.4, phase-wise.

The proposed DGDCT can be found significant in view of the following issues:

1. Embedded cluster Detection,

2. Handling of single linkage problem,

3. Handling of huge datasets (Scalability),

The first two advantages is due to the fact that the clustering algorithm as given in Section 3.4 can identify embedded clusters and can handle the problem of single linkage which is inherent to most of the density based algorithms. DGDCT is scalable to huge datasets as it uses a fast partitioning technique to distribute the huge
data to different nodes for local clustering and finally merges the cluster results at the initiator. The actual clustering is done on the distributed data and hence the processing load reduces even in case of huge datasets. This has been explained in detail in Section 5.3.1.

An overview of the hardware architecture is shown in Figure 5.4. It consists of a number of nodes (e.g. PCs) connected via a network (e.g. Ethernet).
5.3.1 Phase I: Partitioning the dataset

Phase I of the architecture is executed in one of the nodes (initiator node). The dataset is spatially divided into equal sized square grid cells and density of each grid cell is computed. The square mesh is then partitioned with some overlap between adjacent partitions and distributed over \( k_p \) available computers (nodes). No subsequent movement of data between partitions will take place.

Initially, the data space is divided into \( gr_n \times gr_n \) non-overlapping square grid cells, where \( gr_n \) is a user input, and maps the data points to each cell. It then calculates the density of each cell. Assuming, the grid mesh \( D^* \) contains the set of \( gr_n \times gr_n \) objects say, \( D^* = O_0, O_1, O_2, \cdots, O_{(gr_n \times gr_n)-1} \). Suppose, \( O_j = (a_{o_j}, a_{1j}, a_{2j}, \cdots, a_{(n-1)j}, d_n) \) represents a grid cell with \( n \) real-valued attributes \( a_i, i = 0, \cdots, n - 1 \) and density \( d_n \). The \( t^{th} \) attribute value of object \( O_j \) is drawn from domain \( a_j \). If there are \( k_p \) clients, the grid mesh \( D^* \) is partitioned into \( k_p \) subsets \( D_0, D_1, \cdots, D_{k_p-1} \) ordered in sequence. We refer the clients by the corresponding partition \( D_j \) that it receives for processing.

\[
D^* = D_0 \cup D_1 \cup D_2 \cup \cdots \cup D_{k_p-1}
\]

\[
D_i \cap D_j = \emptyset; i, j = 0, 1, \cdots, k_p - 1
\]

The partially overlapped partitions are shown in Figure 5.5 for 2D case. An overlap of one grid cell occurs between two adjacent partitions. The overlapped regions are much smaller than the partitions. The grid cells in the overlapped regions are locally clustered in both the adjacent partitions. Thus they provide the information for merging together the local clustering results of two adjacent partitions. The overlapped width should be at least one cell width because adjacent cells are neighbors according to Definition 3. The grid mesh \( D^* \) is partitioned in this manner based on the values of a selected attribute of the data objects say \( a_s \) as in [BBD04]. The values of \( a_s \) have a range of \([min_{a_s}, max_{a_s}]\). We need to select \((k_p + 1)\) constants in the given range. Let \( c^*_i, i = 1, \cdots, k_p + 1 \) represents the constants such that \( c^*_1 = min_{a_s}, c^*_{k_p+1} = max_{a_s} \) and \( c^*_i < c^*_{i+1} \). Therefore the overlapped region can be represented as:

\[
D_i = \exists j(O_j \in D^*) | c^*_i - cell\_width \leq a_{s_j} \leq c^*_i + 1, i = 2, \cdots, k_p - 1
\]
Figure 5.5: Overlapped spatial partitioning of a 2D dataset

Figure 5.6: Here the dataset is divided into three partitions and transmitted to three computers \((N_{d_p})\) for local clustering, \(k_p = 1, 2, 3\)
\[ D_i = \exists (O_j \in D^*) \mid c_i^s \leq a_{sj} \leq c_{i+1}^s + \text{cell.width}, i = 1 \]

\[ D_i = \exists j (O_j \in D^*) \mid c_i^s - \text{cell.width} \leq a_{sj} \leq c_{i+1}^s, i = 1 \]

The constant \( c_i^s \) should be selected in such a manner that \(| D_j | \) becomes nearly equal to \([N/k_p]\), where \( N \) is total number of data points in the dataset. Moreover, those grid cells which fall within the overlapped regions are marked. Care has been taken for load balancing. The \( k_p \) partitions thus obtained are then sent to \( k_p \) nodes for global as well as intrinsic cluster detection (Figure 5.6).

A detailed discussion on the basic sequential algorithm i.e. GDCT is already reported in Section 3.4, however, it was not scalable to huge datasets.

**Load Balancing**

Partition \( D_i \) is sent to processor \( P_i, i = 1, 2, \ldots, k_p \) for concurrent clustering. Since no data movement takes place after the partitions are received by the respective nodes, care should be taken so that each processor receives nearly equal number of data objects for processing. This will ensure that all the processors finish the clustering job at the same time provided the processors have same processing speed. If the processing speeds are different, then the input data should be distributed to the processors proportionate to their processing speed. We assume that the speeds of the processors are nearly equal and they receive nearly equal amount of data. For doing this, the range of \( a_s \) is divided into intervals of width of one cell-width and the frequencies of data in each interval is counted. Let \( b = [(\max. a_s - \min. a_s)/\text{cell.width}] \), \( N' = [N/k_p] \), \( d_1 = \min. a_s \),

\[ d_i = d_{i-1} + \text{cell.width}, i = 2, 3, \ldots, b \]

\[ F_i = \exists j (O_j \in D^*) \mid d_i \leq a_{sj} \leq d_{i+1}, i = 2, 3, \ldots, b \]

\[ f_i = | F_i | \]

Now, the constants, \( c_i^s \) defined earlier, are computed as \( c_i^s = d_s \) such that \( \sum_{j=1}^s f_j \leq N' \leq \sum_{j=1}^{s+1} f_j, i = 1, 2, \ldots, k_p \), which will ensure that each partition gets number of objects nearly equal to \( N/k_p \).
Minimized communication cost

The proposed method saves transmission cost by avoiding inter-node communication during the process of local clustering. To achieve this goal, each concurrent process of GDCT in each of the nodes, $N_d = 1, 2, \ldots, k_p$, should avoid accessing those data located on any of the other computers, because the access of the remote data requires some form of communication. Therefore, nearby objects should be available on the same computer. This is why an overlap of one cell width has been taken into consideration.

5.3.2 Phase II: Local Clustering

Phase II of the architecture is executed in each of the $k_p$ nodes. This phase plays the actual role of clustering. In this phase, each node executes the GDCT algorithm over the partition of data received from the initiator node to detect the global and nested clusters.

For the partition $D_i$ in node $i$, the grid cells in it will be assigned cluster id according to the clusters formed in that partition. The cluster ids will be used during the server based merging process by the initiator node.

The cluster expansion based on grid cells helps to achieve a significant cost reduction as all the data points are not considered for cluster expansion only the density information of each cell is used. Also, as the cluster id information are used during Phase III merging process, it saves the cost of merging to a great extent.

5.3.3 Phase III: Merging

In Phase III, the cluster results received from the $k_p$ nodes undergo a simplified, yet faster merging procedure to obtain the final clusters. Since the Phase II process in a node may yield more than one cluster along with the embedded clusters, so there are always possibilities for merging during Phase III operation. The Merger module works as follows:
1. Join the partitions received from the $k_p$ nodes according to their overlapping marks.

2. Consider the marked grid cells (overlapping cells) of the candidate clusters.

   2.1 If any of the marked grid cells is identified by different cluster IDs by different partitions (say $l, m$), then assign any one of the IDs (say $l$) to that cell.

3. Assign all those cells having the same cluster ID as the replaced ID ($m$) with $l$.

### 5.3.4 Complexity Analysis

**Phase I:** The partitioning of the dataset into $gr_n \times gr_n$ non-overlapping cells results in a complexity of $O(N)$ where $N$ is the total number of data points. The grid mesh $D^*$ is spatially partitioned into $k_p$ partitions with overlap of one cell width which results in a complexity of $O(gr_n \times gr_n)$, where $gr_n << N$. Each of these $k_p$ partitions will have nearly equal (approximately $N/k_p$) data points. The data points along with the grid information for each of $k_p$ partitions will be sent to the $k_p$ nodes. Therefore $(N/k_p) + t$ points will be sent, where $t$ is the average number of points present in an overlapped region. Next, to transmit these $(N/k_p) + t$ points to each node requires a communication time of $O((N/k_p) + t)$.

**Phase II:** This phase is executed in each of the $k_p$ nodes. Computing density of the cells in each node requires $O((gr_n \times r) \times ((N/k_p) + t))$, where $r$ is the average number of cells along the selected attribute based on which partitioning in Phase I has been performed. The sorting of cells according to their density results in a complexity of $O((gr_n \times r) \log(gr_n \times r))$.

The expansion of the coarse cluster results in $O(m_c)$ time complexity, where $m_c$ is the number of cells in an coarse cluster formed and $m_c << (gr_n \times r)/k_p$ in the average case. Cell subdivision into triangles takes place only in case of the border cells of the coarse cluster and its neighboring cells, Say, there are $p$ border and $q$ neighbor cells where $q >> p$. This step results in a complexity of $O(p + q)$. If the
number of clusters obtained is k then the overall time complexity for the clustering
will be $O(k \times m_c \times (p + q))$.

Therefore, total time complexity will be $O((gr_n \times r) \times ((N/k_p) + t)) + O((gr_n \times r) log(gr_n \times r)) + O(k \times m_c \times (p + q))$. Thus the complexity due to density calculation almost dominates the other components, since $(N/k_p) + t >> (gr_n \times r)$. The clusters detected in this phase are transmitted back to the initiator node with a transmission cost of $O((N/k_p) + t))$.

Phase III: Merging of the clusters obtained from the $k_p$ nodes will take $O(N + k_p.t)$ time.

Thus, the overall time complexity of distributed GDCT will be $O(N) + O(gr_n \times gr_n) + O((N/k_p) + t)) + O((gr_n \times r) \times ((N/k_p) + t)) + O((N/k_p) + t)) + O(N + k_p.t)$. Therefore, the time complexity of DGDCT becomes $O(N)$ since $N >> (gr_n \times gr_n)$.

5.3.5 Performance Evaluation

This section reports an empirical study of DGDCT by measuring execution time, speedup, efficiency and scale-up factors. Since there is no inter-processor communication except for a single processor communicating with each of the remaining processors. Each processor has the same specification i.e. PIV with 1 GHz speed and 128 MB RAM and the processors are connected through Ethernet LAN of speed 10/100 Mbps. measurements. Our implementation is in C in Linux environment and we considered several synthetic datasets containing arbitrary number of arbitrary shaped clusters having $2 \times 10^5$, $4 \times 10^5$, $6 \times 10^5$ and $9 \times 10^5$ objects respectively and experimentation was carried out.

The graph of Parallel Execution Time is shown in Figure 5.7. From the graph we conclude that the execution time decreases significantly as the number of processors increases.

The Relative Speedup curves for two datasets with points $N = 9 \times 10^5$ and $6 \times 10^5$ is
The number of dimensions and the number of clusters are fixed for both the datasets.

The scale-up characteristic of the DGDCT has been found to be satisfactory with the increase in the number of processors as can be seen from Figure 5.9. Here, the number of data points is scaled by the number of processors while dimensions and number of clusters are held constant. It is seen from the Figure 5.10 that if too many processors are used then performance degrades.

DGDCT is an effective technique for handling huge 2D numeric datasets qualitatively. However, DGDCT can be applied only to 2D spatial data. For higher dimensional spatial data, DGDCT may be modified in the line of [AGGR98]. For clustering high resolution massive satellite data, a grid density based clustering technique based on DGDCT is reported in the next section.
Figure 5.8: Relation between Speedup and number of processors for two datasets.

Figure 5.9: Scale-up curve.
5.4 Distributed Grid-Density based Clustering Technique for Satellite Data (DisClus)

Clustering conserves the homogeneous property within a cluster i.e., data points within a cluster are more similar than the data points belonging to different clusters [HK06]. A high resolution satellite image is a remotely sensed image of the earth's surface which is a collection of huge amount of information in terms of number of pixels where each pixel in the image represents an area on the earth's surface. Multi-spectral images are the main type of images acquired by remote sensing. This technology was originally developed for space-based imaging which can capture light of frequencies beyond the visible range of light, such as infrared, which helps to extract additional information that the human eye fails to capture with its receptors for red, green and blue. A multi-spectral satellite image is a digital image comprising of multiple bands where each band represents a particular wavelength of light. Remotely sensed satellite images mainly consists of objects (regions) such as vegetation, water bodies, concrete structures, open spaces, habitation, clouds etc. which are separated due to their different reflectance characteristics, leading to wide variety
of clusters of different sizes, shapes and densities.

Based on our selected survey and experimental analysis, it has been observed that handling large scale data is a challenging task. To discover clusters of varying shapes and sizes effectively over massive spatial datasets is a difficult task. To address these challenges, this chapter presents a distributed grid-density based clustering algorithm (DisClus\textsuperscript{5}) based on SATCLUS and GDSDC [SB10] which can detect clusters over high resolution satellite datasets qualitatively. Further, the post processing phase helps in smoothening the bordering regions of clusters. The method was tested and evaluated over satellite datasets and the results has been found satisfactory.

\textbf{5.4.1 The Proposed DisClus}

Like, DGDCT, DisClus also works in three phases. In the first phase, the satellite image is partitioned into regions with marked overlappings at an initiator node and sent to each of the nodes available for clustering. The second phase is executed in each of the participating nodes. In this phase, the clustering of the data for each partition is performed using either one of the techniques, SATCLUS or GDSDC, at each node. Finally, during the third phase, the nodes transmit the cluster results back to the initiator node where the result are merged to get the final result.

The proposed architecture adopts a shared nothing architecture. It considers a system having $k_p$-nodes where the whole image $D$ is located in any of the nodes (say \textit{node 1}, also referred here as \textit{initiator node}). It executes a fast partitioning technique to generate the $k_p$ initial overlapped partitions. The partitions are then distributed among $k_p - 1$ nodes and one partition is kept at the initiator for cluster detection. Finally, the local cluster results are received from the nodes at this node (\textit{node 1}) and a merger module is used to obtain the final cluster results. Next, each of these phases is explained in brief.

\textbf{Phase I:} In the \textit{initiator node}, the dataset is spatially divided into $gr_n \times gr_n$ non-overlapping square grid cells, where $gr_n$ is a user input, and maps the data points

\footnote{This work is an outcome of a research project funded by ISRO under RESPOND scheme}
to each cell. It then calculates the density of each cell. The grid mesh is then partitioned with some overlap between adjacent partitions and distributed over \( k_p \) available computers (nodes). No subsequent movement of data between partitions will take place. An overlap of single grid cell width occurs between two adjacent partitions. The grid cells in the overlapped regions are locally clustered in both the adjacent partitions. Thus, they provide the information for merging together the local clustering results of two adjacent partitions.

**Load Balancing:** Partition \( D_i \) is sent to processor \( P_i, i=1,\cdots, k \) for concurrent clustering. Since no data movement takes place after the partitions are created and transmitted to the respective nodes till the clustering results are locally available at each node, care has been taken so that each processor receives nearly equal number of data objects (i.e., pixels) for processing. Like DGDCT, here also it is assumed that the speed of all the processors are equal. The range of \( a_s \) is divided into intervals of width of one \( \text{cell.width} \) and the frequencies of data in each interval is counted. The load balancing is done in a manner similar to [BBD04] which ensures that each partition gets number of objects nearly equal to \( N/k_p \).

**Phase II:** In this phase, either SATCLUS or GDSDC (discussed in previous chapter) is executed in each of the \( k_p \) nodes over the partition of data received from the initiator node. For the partition \( D_i \) in node \( i \), the grid cells in it will be assigned \( \text{cluster.id} \) according to the clusters formed in that partition.

The cluster expansion based on grid cells reduces the computation time as data points are not considered for cluster expansion, only the density information of each cell is used. Moreover, the information of the marked cells used during merging process of Phase III saves the cost of merging to a great extent. Finally, Phase II transmits the cluster objects to the initiator node along with the \( \text{cluster.id} \).

**Phase III:** Here, the cluster results are gathered from the \( k_p \) nodes into the initiator node. A merger module is used which uses the \( \text{cluster.id} \) information obtained from the partitions to finalize the cluster results. The Merger module first joins the
partitions received from the $k_p$ nodes according to their overlapping marked cells. It considers the marked grid cells (overlapping cells) of the candidate partitions. If any of the marked grid cells is identified by different \textit{cluster\_ids} by different partitions (say $l$, $m$), then the smallest of the \textit{cluster\_ids} (say $l$) is assigned to that cell. Finally, all those cells having the same \textit{cluster\_id} as that of the replaced \textit{cluster\_id} ($m$) is assigned with \textit{cluster\_id} $l$.

The following lemma provides the theoretical basis for the merging process.

**Lemma 6.** Let $m$ be a marked cell in the overlapping region of two adjacent partitions $p_i$ and $p_{i+1}$ and $C_i$ and $C_j$ are two clusters belonging to $p_i$ and $p_{i+1}$ respectively. If $m \in C_i$ and also $m \in C_j$, then $C_i$ and $C_j$ are merged.

**Proof.** Suppose, $m$ be a marked cell and cell $x \in C_i$ in $p_i$ and cell $y \in C_j$ in $p_{i+1}$. If $m \in C_i$ and also $m \in C_j$, then $x$ and $y$ are reachable from $m$ and $m \in C_i \cap C_j$. So, $x$ is connected to $y$ and cells $x$ and $y$ should be in the same cluster. Therefore, clusters $C_i$ and $C_j$ should be merged. \hfill $\square$

### 5.4.2 Complexity Analysis

Since the proposed technique is executed in three phases and each phase is independent of each other, therefore, the total complexity will be the sum of the complexities due to these three phases.

The first phase divides the dataset of $N$ points into $gr_n \times gr_n$ cells which are partitioned into $k_p$ overlapped partitions with a total of $((k_p - 1) \times gr_n)$ overlapped cells. Therefore, this phase results in a complexity of $O(gr_n \times gr_n)$ approximately, where $gr_n \ll N$. After partitioning, $(N + (k_p - 1) \times t)$ points will be transmitted to $k_p$ nodes, where $t$ is the average number of points present in an overlapped region, results in a complexity of $O((N + (k_p - 1) \times t))$.

The second phase results in a complexity of $O(((gr_n \times gr_n)/k_p + gr_n) + (C^l \times b))$ [SB10], where $C^l$ is the number of clusters detected locally and $b$ is the number of border points obtained in a partition in a node. The clustered points are re-transmitted
to the initiator node with a transmission cost of $O((N + (k_p - 1) \times t))$.

The third phase is responsible for merging of the clusters resulting in atmost $O(N + k_p \times t)$ time.

Thus, the overall time complexity of DisClus will be $O(gr_n \times gr_n) + O(N + (k_p - 1) \times t) + O(((gr_n \times gr_n)/k_p + gr_n) + (C^i \times b)) + O(N + (k_p - 1) \times t) + O(N + k_p \times t)$. Therefore, the time complexity becomes $O(N)$, since $N \gg (gr_n \times gr_n)$ and also $N \gg ((k_p - 1) \times t)$.

5.4.3 Performance Evaluation

In this section we evaluate the performance of DisClus in light of several real-life satellite image data.

Environment Used

The algorithm was implemented using Java in Windows environment with Pentium IV processor with 1 GHz speed and 256 MB RAM. To smooth out any variation, each experiment was carried out for several times and the average result was taken.

Datasets Used

The algorithm was tested over several real-life satellite images as shown in Table 5.1. The Dataset 1 is shown in Figure 5.11. The clusters obtained from the image of Figure 5.11 are shown in Figure 5.12. Figure 5.13 shows Dataset 2. There is a prominent black stretch across the image which is the river Hoogly. The prominent light patch at the bottom right corner is the Salt Lake stadium and the black patches nearby are the fisheries. Two parallel lines at the upper right hand side of the image correspond to the airport runway in the Dumdum airport. Other than these there are several water bodies, roads, open spaces, etc. in the image.

DisClus automatically detects four clusters for this data as observed in Figure 5.14. From our ground knowledge, we can infer that these four clusters correspond to the
Table 5.1: Results of the clustering algorithm over several multi-spectral satellite images

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Dataset</th>
<th>Spectral Bands</th>
<th>Resolution</th>
<th>Clusters Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>Landsat MSS</td>
<td>4</td>
<td>79 m</td>
<td>4 clusters</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>IRS LISS II image of Kolkata, West Bengal</td>
<td>4</td>
<td>36.25 m</td>
<td>4 clusters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset 3</td>
<td>Cartosat-I image of Sonari, Assam Sonari, Assam</td>
<td>4</td>
<td>2.5 m</td>
<td>5 clusters</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>IRS P6 LISS IV image of Borapani, Meghalaya</td>
<td>4</td>
<td>5.8 m</td>
<td>5 clusters</td>
</tr>
</tbody>
</table>

Figure 5.11: Landsat-MSS
Figure 5.12: DisClus output of Figure 5.11

Figure 5.13: IRS Kolkata
classes: Water Bodies (black color), Habitation and City area (deep gray color), Open space (light gray color) and Vegetation (white color). The river Hoogly, stadium, fisheries, city area as well as the airport runway is distinctly discernible in the output image. The predominance of city area on both sides of the river, particularly at the bottom part of the image is also correctly classified which corresponds to the central part of Kolkata city. Figure 5.15 shows the Kolkata image partitioned using FCM algorithm. It can be seen from the result that the river Hoogly and the city area has not been properly classified. These two objects have been classified as belonging to the same class. Similarly, the whole Salt Lake city as a whole has been put into one class. However, some portions such as canals, the Dumdum airport runway, fisheries, etc. have been classified properly.

The experiments on the images presented next is aimed to handle two different types of terrains (plain and hilly) in order to see the variation of classification accuracy. Dataset 3 shows the plain built up area of Sonari in Sibsagar district of Assam (Figure 5.16).

Some characteristic regions in the image are the river Brahmaputra shown in black
Figure 5.15: FCM output

Figure 5.16: Cartosat-1 of Sonari
color and spirally cutting across the middle of the image, roads, agricultural land, human settlements, etc. The DisClus clustering algorithm automatically detects 5 clusters (Figure 5.17 corresponding to river, road, agricultural land, water bodies and human settlements.

The fourth dataset used in this work shows a view of the Borapani area of the state of Meghalaya (Figure 5.18). The characteristic regions in this image are the Deep water (Deep Blue color), Wetlands (light blue color), Vegetation (Red and Pink colors) and Open spaces (White color).

DisClus clustered the image into five classes as shown in Figure 5.19. The resulting image classified the regions as: deep water (dark blue), wetland (sky blue), vegetation (pink), open spaces (white) and pond water (black). It can be seen that the water body at the left hand top corner of the image has been detected which corresponds well to the ground information available.

From the experimental results given above, we can conclude that the technique is highly capable of detecting clusters of all shapes.
Figure 5.18: IRS image of Borapani

Figure 5.19: DisClus output of Figure 5.18
Figure 5.20: Execution time

Figure 5.21: Relative Speedup curves
5.4.4 Performance and Scalability Analysis

In our implementation environment, there is no inter-processor communication except for a single processor communicating with each of the remaining processors. Each processor has the same specification i.e. PIV with 1 GHz speed and 128 MB RAM and the processors are connected through Ethernet LAN of speed 10/100 Mbps. To smooth out any variation, each experiment was carried out for five times and the average results were taken and each reported data point is to be interpreted as an average over five measurements. Our algorithm was implemented in JAVA in Linux environment in a HP xw8600 WS.

i. **Parallel Execution Time**: $T(k_p)$, the parallel execution time of a program is the time required to run the program on $k_p$ nodes in parallel. When $k_p = T(1)$ denotes the sequential run time of a program on a single processor. Figure 5.20 reveals that the execution time decreases significantly with the increase in the number of processors.
ii. Speedup: Speedup is a measure of relative performance between a multiprocessor system and a single processor system, defined as, \( S(k_p) = T(1)/T(k_p) \). On experimenting it has been found that the speedup factor increases with the increase in the number of processors. Figure 5.21 shows relative speedup curves for two datasets with points \( N = 8 \times 10^5 \) and \( 6 \times 10^5 \). The number of dimensions and the number of clusters are fixed for both the datasets. The solid line represents "ideal" linear relative speedup. For each dataset, a dotted line connects observed relative speedups, which is a sub-linear type.

iii. Efficiency: The efficiency of a program on \( k_p \) processors, i.e. \( E(k_p) \) is defined as the ratio of speedup achieved and the number of processors used to achieve it. \[ E(k_p) = \frac{S(k_p)}{k_p} = \frac{T(1)}{k_p T(k_p)} \] In case of the proposed technique we observed that too many processors does not ensure the efficiency.

iv. Scale-up: The scale-up characteristic of the proposed technique has been found to be satisfactory with the increase in the number of processors as can be seen from Figure 5.22. Here the number of data points is scaled by the number of processors while dimensions and number of clusters are held constant.

While comparing to DBSCAN, OPTICS, EnDBSCAN, GDLC and Density-isoline, the proposed DisClus requires only two parameters i.e. the number of grid cells, i.e. \( gr_n \) and threshold \( \alpha \). However, based on our extreme experimental studies, it has been observed that the threshold \( \alpha \) does not vary significantly with different datasets.

5.4.5 Comparison of Cluster Quality of DisClus with its Stand-alone Counterparts

The results of clustering the remote sensing images have been evaluated quantitatively using an index, \( \beta \) as in [PGS00]. Let \( n_i \) be the number of pixels in the \( i^{th} \) cluster \((i = 1, \cdots, c)\), \( X_{ij} \) be the vector (of size \( 3 \times 1 \)) of the HSI values of the \( j^{th} \) pixel \((j = 1, \cdots, n_i\)) for all the images in cluster \( i \), and \( \bar{X}_i \) the mean of \( n_i \) HSI values
of the $i^{th}$ cluster. Then, $\beta$ is defined as [PGS00]:

$$\beta = \frac{\sum_{i=1}^{c} \sum_{j=1}^{n_i} (X_{ij} - \bar{X})^T (X_{ij} - \bar{X})}{\sum_{i=1}^{c} \sum_{j=1}^{n_i} (X_{ij} - X_i)^T (X_{ij} - X_i)}$$

where $n$ is the size of the image and $\bar{X}$ is the mean HSI value of the image. It may be noted that $X_{ij}$, $\bar{X}$, and $X_i$ are all $3 \times 1$ vectors. The above measure is the ratio of the total variation and within-cluster variation and is widely used for feature selection and cluster analysis [MMP02]. For a given image and $c$ (number of clusters) value, the higher the homogeneity within the segmented regions, the higher the $\beta$ value. The proposed DisClus has the highest $\beta$ as can be seen in Table 5.2. DisClus was also compared with its other stand-alone and density based counterparts in terms of general parameters and the result is shown in Table 5.3.

### Table 5.2: Comparison of $\beta$ values for different clustering algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>5.30</td>
<td>7.02</td>
<td>9.88</td>
<td>17.82</td>
<td>12.63</td>
<td>15.31</td>
</tr>
</tbody>
</table>

5.5 Discussion

This chapter presents two clustering techniques: the first one (DGDCT) is for massive 2D spatial data and the second one is for satellite data. DGDCT is based on a grid-density based approach and can detect global as well as embedded clusters qualitatively. Experimental results of DGDCT in terms of scale-up and speedup are reported to establish the superiority of the technique in light of several synthetic datasets.

DisClus is also a grid-density based clustering technique for high-resolution multispectral satellite image. The technique was experimentally evaluated and found capable in detecting the clusters qualitatively. Experimental results establish the efficiency of the technique in light of several satellite images. In DisClus, there is also an option for choosing either the partition based algorithm (SATCLUS) or the fuzzy...
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>No. of parameters</th>
<th>Structure</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>1 ((N))</td>
<td>Spherical</td>
<td>(O(N))</td>
</tr>
<tr>
<td>FCM</td>
<td>1 ((N))</td>
<td>Non-Convex</td>
<td>(O(N))</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>2 ((\text{MinPts, } \varepsilon))</td>
<td>Arbitrary</td>
<td>(O(N \log N)) using (R^*) tree</td>
</tr>
<tr>
<td>OPTICS</td>
<td>3 ((\text{MinPts, } \varepsilon, \varepsilon_t))</td>
<td>Arbitrary</td>
<td>(O(N \log N)) using (R^*) tree</td>
</tr>
<tr>
<td>SATCLUS</td>
<td>2 ((gr_n, \alpha))</td>
<td>Arbitrary</td>
<td>(O(k \times r))</td>
</tr>
<tr>
<td>GDSDC</td>
<td>2 ((gr_n, \alpha))</td>
<td>Arbitrary</td>
<td>(O(k \times r))</td>
</tr>
<tr>
<td>DisClus</td>
<td>2 ((gr_n, \alpha))</td>
<td>Arbitrary</td>
<td>(O(N))</td>
</tr>
</tbody>
</table>
based one (GDSDC) for the clustering process depending on the image data. Results of both the algorithms have been reported in Chapter 4 to show their efficiencies. Since satellite images are huge in size, DisClus helps in handling such data efficiently and qualitatively. Moreover, DisClus uses the number of grid cells and threshold $\alpha$ as input parameters, however, it has been seen that $\alpha$ does not vary much with different datasets. The next chapter deals with the application of clustering over gene expression datasets.