JCST Journal of Computer Science & Technology

Indexed in: SCIE, EI, INSPEC, CBST, DBLP, etc.

Sponsored by: ICT, CAS & CCF

A Survey and Experimental Review on Data Distribution Strategies for Parallel Spatial Clustering Algorithms

Challa Jagat Sesh, Goyal Navneet, Sharma Amogh, Sreekumar Nikhil, Balasubramaniam Sundar, Goyal Poonam

View online: http://doi.org/10.1007/s11390-024-2700-0

Articles you may be interested in

Impacts of Dirty Data on Classification and Clustering Models: An Experimental Evaluation

Zhi-Xin Qi, Hong-Zhi Wang, An-Jie Wang

Journal of Computer Science and Technology. 2021, 36(4): 806-821 http://doi.org/10.1007/s11390-021-1344-6

Revisiting the Parallel Strategy for DOACROSS Loops

Song Liu, Yuan-Zhen Cui, Nian-Jun Zou, Wen-Hao Zhu, Dong Zhang, Wei-Guo Wu

Journal of Computer Science and Technology. 2019, 34(2): 456-475 http://doi.org/10.1007/s11390-019-1919-7

GAEBic: A Novel Biclustering Analysis Method for miRNA-Targeted Gene Data Based on Graph Autoencoder

Li Wang, Hao Zhang, Hao-Wu Chang, Qing-Ming Qin, Bo-Rui Zhang, Xue-Qing Li, Tian-Heng Zhao, Tian-Yue Zhang

Journal of Computer Science and Technology. 2021, 36(2): 299-309 http://doi.org/10.1007/s11390-021-0804-3

Semi-Supervised Classification of Data Streams by BIRCH Ensemble and Local Structure Mapping

Yi-Min Wen, Shuai Liu

Journal of Computer Science and Technology. 2020, 35(2): 295-304 http://doi.org/10.1007/s11390-020-9999-y

Scalable and Adaptive Joins for Trajectory Data in Distributed Stream System

Jun-Hua Fang, Peng-Peng Zhao, An Liu, Zhi-Xu Li, Lei Zhao

Journal of Computer Science and Technology. 2019, 34(4): 747-761 http://doi.org/10.1007/s11390-019-1940-x

A GPU-Accelerated In-Memory Metadata Management Scheme for Large-Scale Parallel File Systems

Zhi-Guang Chen, Yu-Bo Liu, Yong-Feng Wang, Yu-Tong Lu

Journal of Computer Science and Technology. 2021, 36(1): 44-55 http://doi.org/10.1007/s11390-020-0783-9

JCST Official WeChat Account

JCST WeChat Service Account

JCST Homepage: https://jcst.ict.ac.cn
SPRINGER Homepage: https://www.springer.com/journal/11390
E-mail: jcst@ict.ac.cn
Online Submission: https://mc03.manuscriptcentral.com/jcst
Twitter: JCST_Journal
LinkedIn: Journal of Computer Science and Technology

Challa JS, Goyal N, Sharma A *et al.* A survey and experimental review on data distribution strategies for parallel spatial clustering algorithms. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(3): 610-636 May 2024. DOI: 10.1007/s11390-024-2700-0

A Survey and Experimental Review on Data Distribution Strategies for Parallel Spatial Clustering Algorithms

Jagat Sesh Challa[1], Navneet Goyal[1], Amogh Sharma[2], Nikhil Sreekumar[3], Sundar Balasubramaniam[1] and Poonam Goyal[1, *]

[1] *Advanced Data Analytics and Parallel Technologies Laboratory, Birla Institute of Technology and Science Pilani 333031, India*

[2] *Uber, New York 11101, U.S.A.*

[3] *Computer Science and Engineering Department, University of Minnesota, Minneapolis 55455, U.S.A.*

E-mail: jagatsesh@pilani.bits-pilani.ac.in; goel@pilani.bits-pilani.ac.in; amogh@uber.com; sreek012@umn.edu sundar.b@wilp.bits-pilani.ac.in; poonam@pilani.bits-pilani.ac.in

Received August 24, 2022; accepted March 13, 2024.

Abstract The advent of Big Data has led to the rapid growth in the usage of parallel clustering algorithms that work over distributed computing frameworks such as MPI, MapReduce, and Spark. An important step for any parallel clustering algorithm is the distribution of data amongst the cluster nodes. This step governs the methodology and performance of the entire algorithm. Researchers typically use random, or a spatial/geometric distribution strategy like *kd*-tree based partitioning and grid-based partitioning, as per the requirements of the algorithm. However, these strategies are generic and are not tailor-made for any specific parallel clustering algorithm. In this paper, we give a very comprehensive literature survey of MPI-based parallel clustering algorithms with special reference to the specific data distribution strategies they employ. We also propose three new data distribution strategies namely Parameterized Dimensional Split for parallel density-based clustering algorithms like DBSCAN and OPTICS, Cell-Based Dimensional Split for dGridSLINK, which is a grid-based hierarchical clustering algorithm that exhibits efficiency for disjoint spatial distribution, and Projection-Based Split, which is a generic distribution strategy. All of these preserve spatial locality, achieve disjoint partitioning, and ensure good data load balancing. The experimental analysis shows the benefits of using the proposed data distribution strategies for algorithms they are designed for, based on which we give appropriate recommendations for their usage.

Keywords parallel data mining, data distribution, parallel clustering, spatial locality preservation

1 Introduction

Data clustering is one of the most commonly used data mining techniques for knowledge discovery. Clustering partitions the data into meaningful groups, known as clusters, such that the dissimilarity between the objects belonging to the same cluster is minimized and the dissimilarity between the objects from different clusters is maximized[1]. Clustering algorithms are classified into multiple categories. Algorithms such as k-means[2], K-medoids[3], Bisecting k-means[4], and so on, fall in the category of partitioning-based clustering. Algorithms such as DBSCAN[5], OPTICS[6], SNN[7], DENCLUE[8], and so on, fall in the category of density-based clustering. Algorithms such as SLINK[9], CLINK[1], Average LINK[1], and so on, fall in the category of hierarchical clustering. Algorithms such as CLIQUE[10], MAFIA[11], ENCLUS[12], PROCLUS[13], ORCLUS[14], FINDIT[15], and so on, fall in the category of subspace clustering. Algorithms such as STING[16], CLIQUE[10], MAFIA[11], and so on, fall in the category of grid-based clustering. These algorithms are used in many applications such as satellite image segmentation[17], noise filtering and outlier detection[18], bio-informatics[19], prediction of stock prices[20], and so on.

Survey

*Corresponding Author

The advent of big data systems has led to generation of data at a faster pace and at a cheaper cost, creating a data deluge. To discover knowledge from such data, parallel clustering algorithms have been proposed to work over distributed memory architectures. A few such solutions include parallel partitioning-based clustering algorithms[21–23], parallel density-based clustering algorithms[24–30], parallel subspace clustering algorithms[31–33], parallel hierarchical clustering algorithms[34, 35], etc. These solutions are typically based on parallel frameworks such as MPI, MapReduce or Spark, and are executed over a distributed memory architecture, which typically comprises a cluster of computing nodes. A few of the above approaches also work for shared memory and GP-GPU-based architectures. All of the above parallel algorithms are data parallel, i.e., data is distributed to the processors using a suitable partitioning technique and the same set of instructions/code is executed over each partition.

A typical data-parallel clustering algorithm has its workflow as the following steps.

- *Step* 1—*Data Distribution*. In this step, the data is distributed amongst all the computing nodes of a cluster. The kind of distribution used depends upon the design and requirements of the algorithm. The distribution of data enables each computing node to process the data chunk allocated to it in parallel. Data load balancing is an important criterion to be considered while distributing the data for parallel algorithms. Commonly used data distribution strategies are random partitioning and spatial partitioning (e.g., *kd*-tree based partitioning, grid-based partitioning).
- *Step* 2—*Local Computations*. In this step, each computing node executes local computations for the chunk of data allocated to it and produces an intermediate result, e.g., local clustering. This step might require inter-node communication to fetch extra data (required for local computations) from other computing nodes of the cluster depending upon the design of the algorithm. Some algorithms do not require to fetch any data from other computing nodes, some algorithms fetch during the execution of local computations as and when required, and some (recent) algorithms fetch before beginning the local computations step, making minimum inter-node communication possible. For example, parallel k-means[21] does not require data from other computing nodes for local computations. It is an embarrassingly parallel algorithm. The dGridSLINK algorithm[35] is a re-designed parallel SLINK that fetches data during the local computations for merging two clusters. And, parallel DBSCAN fetches data from other computing nodes before beginning the local computations[24, 28], for accurately computing the ϵ-neighborhoods of all the points.
- *Step* 3—*Merging of Local Results*. In this step, the intermediate/local clustering results from all the computing nodes are merged either sequentially or in parallel to give a global clustering result. This step again requires inter-node communication. An algorithm may iterate over step 2 and step 3 if required by its design to obtain final clustering (e.g., dGridSLINK[35]).

In the above workflow, data distribution plays a pivotal role in reducing the cost of inter-node communication incurred in step 2 and step 3, thus influencing the overall performance of the algorithm. Some parallel algorithms simply use random distribution[34, 36, 37] and some parallel algorithms that execute spatial queries (like ϵ-neighborhood and k-nearest neighbor queries) use a distribution that preserves spatial locality of the data points either partially or fully[24, 28, 35, 38–40]. By preservation of spatial locality, we mean that for a given data point p, the data points neighboring p are available locally in the same computing node. Spatial locality helps in reducing the overlap in the search space of different computing nodes. Fig.1 shows three different kinds of data distributions. Each part of the figure shows a spatial arrangement of data points that are distributed to four different computing nodes, A, B, C, and D. Fig.1(a) depicts a random distribution where the search space of all the nodes overlap. In Fig.1(b), the data points are somewhat spatially organized with some overlap in their search space. In Fig.1(c), the data points are organized in a perfectly disjoint manner with zero overlap in the search space of the computing nodes.

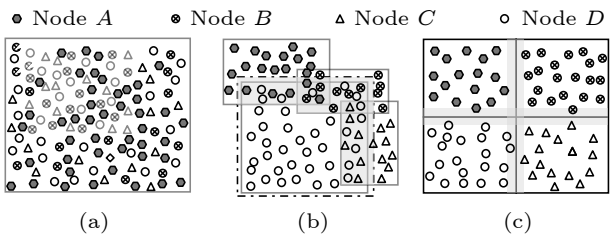


Fig.1. Spatial locality preservation. (a) None for random distribution. (b) Moderate for partially disjoint. (c) High for fully disjoint[41].

Consider the case of parallel DBSCAN[24, 28]. In

the local computations step, the algorithm executes ϵ-neighborhood queries for all the points in every computing node. To compute ϵ- neighborhood correctly for all the points in it, we shall require to fetch extra data points from all the other computing nodes in the case of random distribution in Fig.1(a), as ϵ-boundaries of points $\in A$ are overlapping with the search spaces of all the other computing nodes. We can see that no spatial locality is preserved in this distribution. For distribution shown in Fig.1(b), a lesser number points shall be needed to be retrieved from other nodes for performing ϵ-neighborhood queries for all the points $\in A$ as the overlap is lesser, and still fewer points in the case of disjoint distribution shown in Fig.1(c). This is because spatial locality is best preserved in the last case. Thus, a good spatial distribution helps reduce inter-node communication of step 2 and step 3, thus reducing the overall computation cost of the underlying algorithm.

Literature reveals only very few distribution strategies used in parallel clustering algorithms. They include random partitioning[34, 36, 37], *kd*-tree partitioning[24, 28, 38, 39] and grid/cell-based partitioning[27, 35, 40, 42].

1.1 Research Gap and Motivation

Although the above distribution schemes are being used for parallel data mining algorithms, they are not specifically designed for such use. Also, they do not capture any specific data access patterns associated with the clustering algorithm (or a class of algorithms). Tailor-made partitioning strategies can be designed for specific parallel clustering algorithms that specifically capture their respective design requirements.

Also, there is no specific work reported in the literature that addresses or reviews the data distribution problem, describes the current approaches, or presents new distribution techniques.

1.2 Our Contributions

- In this paper, we give a very comprehensive survey of MPI-based parallel clustering algorithms, specifically discussing their workflow and the data distribution strategies they employ. To the best of our knowledge, this paper presents the first such discussion.
- Then, we propose three new spatial data distribution strategies: Parameterized Dimensional Split (PD-Split), Cell-Based Dimensional Split (CD-Split), and Projection-Based Split (Pbased-Split). These strategies are variations of *kd*-tree based partitioning.
- We also demonstrate how these data distribution strategies, including the existing and the proposed three strategies, can be used to distribute very large datasets within the constraints of limited hardware resources. We describe approximate variants of the above strategies that use sampling to mitigate the issue of limited hardware resources.
- We present experimental analysis on the applicability of the above data distribution strategies over the existing parallel spatial clustering algorithms for various large and high-dimensional datasets and give recommendations for their appropriate usage.

The rest of the paper is organized as follows. Section 2 gives a comprehensive literature review of MPI-based parallel clustering algorithms with specific reference to the data distribution strategies they use. Section 3 explains various data distribution strategies proposed in this work. Section 4 gives experimental results and analysis. Section 5 summarizes the inferences drawn from experimentation and gives appropriate recommendations for the usage of the proposed data distribution strategies. Section 6 concludes this paper and gives recommendations for future work.

2 Survey of MPI-Based Parallel Clustering Algorithms

In this section, we give a comprehensive survey of MPI-based parallel clustering algorithms and a brief insight into their data distribution strategies. Table 1 summarizes the data distribution strategies used by each parallel algorithm described in this section.

2.1 Parallel Partitioning-Based Clustering

Partitioning-based clustering algorithms are those algorithms that create k partitions of the data (called clusters) while reducing the inter-cluster similarity and increasing intra-cluster similarity of the data points of each cluster. The clustering is performed iteratively until it converges to the expected thresholds on the distances specified above. The most basic partitioning-based clustering algorithm is k-means[2]. It starts with k randomly picked seeds as centroids and assigns all the points in the dataset to their nearest centroids. Then, the centroids are updated with re-

Table 1. Data Distribution Strategies Used by Various MPI-Based Parallel Clustering Algorithms

Algorithm	Type	Year	*kd*-Tree	Random	Others
Dhillon *et al.*[43]	Partitioning-based	2002		✔	
MKmeans[21, 44]	Partitioning-based	2011		✔	
Kumar *et al.*[45]	Partitioning-based	2011		✔	
Kerdprasop *et al.*[46]	Partitioning-based	2012		✔	
Balcan *et al.*[47]	Partitioning-based	2013		✔	
Gursoy *et al.*[48]	Partitioning-based	2004			Quad-tree based
Di Fatta *et al.*[49]	Partitioning-based	2010	✔		
Kumari *et al.*[22]	Partitioning-based	2015		✔	
Arbelaez *et al.*[50]	Partitioning-based	2013			Space-filling curves
PBKP[51]	Partitioning-based	2007		✔	
PDBSCAN[52]	Density-based	1999		✔	
Zhou *et al.*[53]	Density-based	2000		✔	
Arlia *et al.*[54]	Density-based	2001		✔	
Coppola *et al.*[55]	Density-based	2002		✔	
Brecheisen *et al.*[56]	Density-based	2006			OPTICS-based
P-DBSCAN[57]	Density-based	2010			Projection-based
PDSDBSCAN-D[24]	Density-based	2012	✔		
Pardicle[25]	Density-based	2014	✔		
BD-CATS[26]	Density-based	2015	✔		
HPDBSCAN[27]	Density-based	2015			Grid-based
GridDBSCAN-D[28]	Density-based	2017	✔		
DBSCAN-MS[58]	Density-based	2019	✔		
POpticsD[38]	Density-based	2013		✔	
DOPTICS[39]	Density-based	2015	✔		
dR-SNN[40]	Density-based	2016	✔		
CLUMP[37]	Hierarchical	2009		✔	
Rajasekharan *et al.*[59]	Hierarchical	2005			Grid-based
pPoP[60]	Hierarchical	2007			Partially overlapping partitioning
PINK[34]	Hierarchical	2013		✔	
GridSLINK[35]	Hierarchical	2016			CD-Split
PMAFIA[61]	Subspace	2000		✔	
Goyal *et al.*[33]	Subspace	2016		✔	

spect to the membership obtained. The above steps are iteratively repeated, wherein we get new centroids after every iteration. The iterations continue until a threshold is reached on the inter-cluster similarity. Variations of k-means clustering include k-median[62], k-medoids[3], bisecting k-means[4], k-modes[1], k-prototype[1], etc. k-median computes median at every iteration instead of mean. k-medoids identifies a new set of medoids, which are actual points in the dataset, by replacing a medoid with a non-medoid point to reduce the cost. This makes it more robust to noise and outliers than k-means. Bisecting k-means is a hybrid of divisive hierarchical clustering and k-means. k-mode is a variant of k-means specifically suitable for clustering categorical data, and k-prototype is a hybrid of k-means and k-modes to work upon data that has a mixture of categorical and numeric attributes.

2.1.1 *Parallel k-Means*

Parallelization of k-means is very straightforward for a distributed memory architecture. Initially, data is randomly distributed to each computing node and a list of initial centroids is chosen randomly. These centroids are known to every computing node of the cluster. Now, in every computing node, the distance between the data points and centroids are calculated and the points are assigned to the nearest centroid. Then a new set of centroids is computed for each node. These sets of new centroids from each node are then averaged into a new set of global centroids, us-

ing which the next iteration begins. In this way, the algorithm goes on until the threshold criterion is met.

A few implementations of MPI-based k-means algorithm that have been proposed include [21, 43–47]. [21, 43, 44] present variants of parallel k-means clustering that have a similar workflow as explained above. [45] uses parallel k-means on massively parallel systems to cluster large remote sensing data. [46] presents approximate parallel k-means (APKM) using sampling-based techniques. [47] presents parallel k-means and parallel k-median algorithms using coresets. All of the parallel k-means variants use random data distribution.

Apart from these, a few variants of parallel k-means have also been presented that use spatial partitioning techniques. The parallel version presented in [48] uses quad-tree based data distribution. The local computations phase uses kd-trees within each computing node to reduce the total number of distance computations of the k-means algorithm. Another approach was presented in [49] that uses kd-tree based data distribution. The workflow of these algorithms remains the same as that of parallel k-means, without any requirement of additional data from other computing nodes. However, these approaches make use of locality achieved in distribution to reduce the overhead in terms of distance computations performed within each node.

A few shared memory based parallel k-means have also been presented[63, 64]. A few implementations have also been presented to work over MapReduce/Hadoop that include [65–67]. A novel data distribution strategy for MapReduce-based parititional clustering algorithms has also been proposed in [68]. This partitioning strategy can be applied to multiple variants of partitioning-based clustering. It attains performance by distributing the dataset and avoids iterative MapReduce jobs. The experiments show that their approach performs better than the state-of-the-art that do not use this distribution strategy. A few approaches have also been proposed for Spark/RDDs[69, 70], which use random distribution.

2.1.2 *Parallel k-Means with Seed Selection*

Initial seed selection is an important decision to improve the overall performance of the k-means algorithm. Various parallel seed selection algorithms have been proposed in the literature. One of them is the Scalable k-means++[71], which has been proposed for the MapReduce framework. A more recent approach was proposed in [22] for MPI-based systems. This approach not only gives an efficient parallel approach for seed selection but also efficiently pre-processes the data in parallel to make the parallel k-means efficient. Both the above approaches typically employ random data distribution. This is because, like k-means, these are also embarrassingly parallel and do not require fetching of extra points from other computing nodes.

2.1.3 *Parallel k-Medoids and Bisecting k-Means*

There is one MPI-based implementation for parallel k-medoids[50] that uses a similar workflow as that of parallel k-means. However, it uses space-filling curves such as the Hilbert Curve and Dimension-Sort Curve for partitioning the data amongst the computing nodes. These curves induce an ordering among the data points, which is used for packing the points into multiple computing nodes of the cluster. The induced ordering achieves the preservation of spatial locality, but not completely disjoint partitioning. Overall, the algorithm's performance improves with the usage of these curves.

A few MapReduce-based implementations of parallel k-medoids clustering include [72, 73]. A few Spark-based implementations include [23, 74]. These approaches usually employ random data distribution.

One variant of parallel bisecting-k-means (PBKP) was proposed in the literature for MPI-based systems[51]. This algorithm also uses random data distribution. Again, the overall workflow remains similar to that of parallel k-means, except that the number of clusters/centroids increases as more bisections are performed.

2.2 Parallel Density-Based Clustering

2.2.1 *Parallel DBSCAN*

DBSCAN is the most commonly used density-based clustering algorithm[5]. It finds clusters of data points with respect to two parameters ϵ (> 0) and min_pts (> 0). It computes ϵ-neighborhood for each point in the dataset and labels the point as *core*, *border*, or *noise*. A core point initiates a cluster and the cluster is expanded by computing neighborhoods for points in the ϵ-neighborhood of the above core point, which is recursively repeated until no core point is found. This completes the expansion of the cluster. The next random point from the remaining unpro-

cessed points is visited to extract another cluster and this process continues until all the points are processed. The time complexity of the DBSCAN algorithm that uses R-tree for neighborhood queries, is $O(N \log N)$, where N is the size of the dataset.

Early approaches to the parallelization of DBSCAN adopt the master-slave model for computation[52–57]. All these approaches simply distribute data to the slave computing nodes randomly, while maintaining data load balancing. The first such approach is PDBSCAN[52], which uses dR*-tree for region queries. dR*-tree is a variant of R*-tree[75] in which R*-tree is replicated over multiple computing nodes for efficient data access on a distributed system. The local clusterings from all the slave nodes are aggregated to form global clustering in the master node. The next approach presented in [53] proposes various optimizations to DBSCAN and proposes a parallel version, which follows a similar workflow as explained above. The next approach given in [54] gives a parallel DBSCAN algorithm in which, DBSCAN is divided into two major operations: clustering assignment and neighborhood querying. The master node performs clustering assignment while all slaves perform neighborhood queries in parallel for their respective partition of data obtained by random partitioning. The next approach proposed in [55] also presents a similar master-slave model-based parallel DBSCAN in which each slave keeps a copy of an R*-tree to execute neighborhood queries. The major drawback of such master-slave approaches is the serialized computation at the master node, which limits their scalability to a larger number of computing nodes. Another client-server model-based parallel DBSCAN has been presented in [56], which uses OPTICS[6] for data partitioning to assign adjacent enumeration values to similar objects. This is computationally expensive as the cost of OPTICS itself is larger than $O(N^2)$. Approximate clustering is obtained as lower-bounding distance values conservatively approximating the exact clustering. The next approach[57] proposes a parallel DBSCAN, named P-DBSCAN, which distributes the data among several nodes, builds Priority R-tree (R-tree variant) on each node, runs local DBSCAN, and aggregates the local results to get global clustering results. It uses a kind of projection-based spatial partitioning, where the data points are projected over coordinate axes and the partitions are created in an order exhibited by the projection (say from left to right). This distribution ensures that the partitions are spatially disjoint.

All the approaches described above incur high communication cost between master and slave nodes. Most of them have a sequential data access pattern. Moreover, the parallelization during the merging phase, in many cases, is limited due to the random partitioning. And most importantly, they do not exploit the spatial locality required by neighborhood queries used in DBSCAN during the data distribution phase. All of the above reasons render the master-slave model inefficient for DBSCAN.

The first approach to DBSCAN that breaks the sequential data access pattern solidly is PDSDBSCAN[24]. It uses union-find (UF) data structure, which also makes it amenable to parallelization and achieves better scalability. The authors presented the parallel versions PDSDBSCAN-D and PDSDBSCAN-S for distributed and shared memory systems, respectively. PDSDBSCAN-D uses *kd*-tree based partitioning for data distribution, giving a completely disjoint partitioning (see Fig.2). After partitioning, local clustering is performed at each computing node, followed by merging of the local clusterings into a global clustering output. Before performing local clustering, each computing node additionally fetches data lying in its ϵ-extended boundary from the other nodes to compute fully accurate ϵ-neighborhoods (see Subsection 3.4 for details). The completely disjoint nature of the *kd*-tree based partitioning makes sure that this extra data fetched is minimal, thereby reducing the communication overhead to a great extent.

A couple of heuristic-based approximate DBSCAN clustering algorithms—Pardicle[25] and BD-CATS[26] were also proposed in the literature, which are

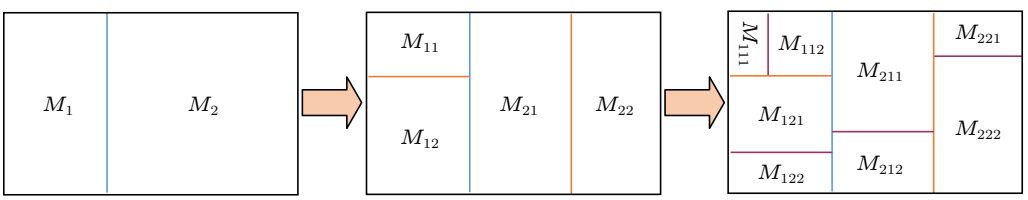


Fig.2. *kd*-tree based data partitioning or KD-Split[24, 28].

based on PDSDBSCAN-D[24]. These two algorithms are capable of processing massive datasets with some approximation in the results. These algorithms use density-based sampling for achieving efficient execution. Both the algorithms claim a high value of omega-index (0.99), which shows that the clustering is close to the actual clustering produced by that of traditional DBSCAN. Both the above algorithms also use *kd*-tree based partitioning for data distribution and have a similar workflow like that of PDSDBSCAN-D.

Recently, a grid-based parallel implementation of DBSCAN, HPDBSCAN[27], has been proposed. This approach uses a grid/cell-based data distribution approach. First, equal-size data is read by all p processors. A grid is overlaid on all the data, and then the cells thus formed are re-distributed among the computing nodes based on load-balancing cost heuristics. The partitioning obtained is completely disjointed. Then local computations are performed on each computing node and then the results are merged into global clustering. Another grid-based DBSCAN, GridDBSCAN[28], has been proposed recently, which reduces the total number of neighbourhood queries as well as the search space for each query while producing exact DBSCAN clustering output. It uses a variant of R-tree known as Grid-R-tree[76] that supports efficient querying of cell-wise neighborhoods. GridDBSCAN is parallelized for distributed memory, shared memory, and hybrid architectures. The distributed memory version, GridDBSCAN-D, also uses *kd*-tree based partitioning for data distribution, performs local GridDBSCAN clustering on each node, and merges the local clusterings in a tree-parallel way. The experimental results claim better scalability and run-time performance than the previous approaches.

Another approach is DBSCAN-MS[58], which is specifically designed to work over general metric spaces rather than conventional Euclidean spaces. This uses *kd*-tree based partitioning for data distribution across multiple nodes, performs local clustering in each of the nodes and then merges the local clusterings to get a global clustering output. In the local clustering phase, this algorithm uses pivot filtering and the sliding window techniques for pruning the search space. The experiments show better efficiency when compared with the state-of-the-art.

Another approach presented in [77] proposes an iterative framework for distributed clustering of attributed graphs using Personalized PageRank (PPR) for distance computation and DBSCAN. The proposed method is based on Blogel, a distributed framework for iterative graph query processing. The paper presents efficient strategies for iteratively updating attribute weights and a game theory based approach to refine clustering results for better effectiveness. The paper also uses certain optimisations to reduce the recomputation and communication costs. The experimental results on real datasets indicate the method's scalability, efficiency, and effectiveness.

Apart from MPI-based solutions, there are a few solutions for GPGPU-based systems, MapReduce/Hadoop, Spark, and hybrid systems. Mr. Scan algorithm[42] was first introduced for GPGPU-based systems. It is an approximate algorithm that uses a tree-parallel approach for merging of local DBSCAN clusterings. This algorithm divides the dataset into cells of size $\epsilon \times \epsilon$ (for 2D) and then packs the cells into the computing nodes while achieving spatially disjoint partitioning. It also gives good load balancing, although not perfect. Then few representative points from each cell are used to perform actual clustering, which results in approximation. Mr. Scan was also implemented on a hybrid (CPU+GPGPU) system. A few other GPU-based parallel DBSCAN algorithms proposed include [78, 79]. They employ random data distribution.

A few MapReduce-based implementations for DBSCAN have also been proposed[80, 81]. Similarly, a few Spark-based implementations have also been proposed[29, 82, 83]. All of these use random data distribution.

Another novel parallel Density Peaks Clustering algorithm has been proposed in [84], which is based on MapReduce paradigm. The proposed LSH-DDP, an approximate algorithm, exploits locality sensitive hashing for partitioning data, performs local computation, and aggregates local results to approximate the final results. Experiments show that it has good speed-up without much compromise on the quality of clustering.

2.2.2 *Parallel OPTICS*

OPTICS (Ordering Points To Identify the Clustering Structure) is a hierarchical density-based clustering algorithm[6]. OPTICS addresses DBSCAN"textquotesingle s major limitation: the problem of detecting meaningful clusters of varying density. OPTICS provides an overview of the cluster structure of a

dataset with respect to density and contains information about every cluster level of the dataset. To do so, OPTICS generates a linear order of points where spatially closest points become neighbors. Additionally, for each point, a spatial distance (known as reachability distance) is computed, which governs the density. Once the order and the reachability distances are computed using ϵ and min_pts, we can query for the clusters at a particular value of ϵ' ($\leqslant \epsilon$).

The first parallel version of OPTICS clustering is POPTICSD[38], which first presents MST-OPTICS, which is a re-engineered version of the original OPTICS algorithm. MST-OPTICS breaks the sequential data access pattern of OPTICS and makes it amenable to parallelization. POPTICSD is its parallelization for distributed memory and uses random data distribution. On each partition, a local MST is constructed in the local computations phase, and all those MSTs are merged into a global MST. The clustering results obtained by this approach are comparable but not exactly the same as those obtained by the classical OPTICS algorithm. This is because the local MSTs constructed for each partition are not fully correct as all the data required for the computation is not locally available due to random partitioning. The deviation from the actual clustering increases with the increase in the number of processing elements. Their experiments establish the algorithm's scalability with increasing processing elements.

DOPTICS is another parallel approach[39] that uses *kd*-tree based data partitioning. The computing nodes store their respective data partitions locally in R-trees[85] over which OPTICS is run locally to obtain local orderings. Each computing node fetches extra data lying in the ϵ-extended boundaries from other computing nodes before performing local OPTICS. This extra data is required for computing fully accurate ϵ-neighborhoods of points lying in the boundaries. Then, the local orderings obtained are hierarchically merged into a global final cluster ordering. The final clustering obtained by DOPTICS is identical to that by classical OPTICS. The experimental results show significant speed-up and scalability with increasing processing cores.

2.2.3 *Parallel Shared Nearest Neighbor Clustering*

Shared Nearest Neighbor Clustering (or SNN) is a density-based clustering algorithm that uses two similarity measures known as SNN-similarity and SNN-density[86]. SNN-similarity for two points is defined as the number of shared neighbors if they are in each other's nearest neighbors lists. SNN-density of a point is the number of points that have SNN-similarity of ϵ or greater to it. SNN clustering uses a DBSCAN-like algorithm applied over the core points (using SNNDensity and a min_pts threshold over it) to identify clusters of an arbitrary size and shape to filter noise/outliers. It is especially suited for high-dimensional data.

To the best of our knowledge, there is the only attempt of parallelization of SNN for MPI-based clusters[40]. It first presents the R-SNN algorithm, which is a modification to the classical SNN algorithm, using R-tree for efficient nearest neighbors computations and is more optimized in terms of memory requirement. Its parallelization is the Parallel R-SNN that uses *kd*-tree partitioning in step 1, then local computations are performed over each partition, and then the local results are merged into a global clustering. The spatial partitioning ensures good load balancing and makes the merging step efficient. The experimental analysis shows the scalability of the proposed approach.

A parallel JP-Clustering algorithm (that uses only SNN-similarity for clustering) for the MapReduce framework has also been proposed[87]. A MapReduce based SNN has also been presented in [88].

2.3 Parallel Hierarchical Clustering

Hierarchical clustering is also one of the popular techniques of clustering. There are two kinds of hierarchical clustering algorithms proposed in the literature.

1) *Top-down, Also Known as Hierarchical Divisive Clustering* (*HDC*). It starts with considering all the points in a single cluster and then recursively splits the clusters until some criterion is met[1], which could be a limit on the inter-cluster distances or the number of clusters.

2) *Bottom-up, Also Known as Hierarchical Agglomerative Clustering* (*HAC*). It starts with considering individual points as clusters and then repeatedly merges the closest pairs of clusters until one of the above criteria is met.

The result of any of the above clustering techniques is a dendrogram, which is a tree-like structure showing the clusters agglomerated at each level (refer to [1] for details). There are many variants of HAC

such as SLINK[9], AverageLINK[1], CLINK[1] and, Genie[89]. These variants differ from each other in terms of the way the proximity distance between a pair of clusters is defined. The most popular and widely used variant is the single-linkage or SLINK algorithm that has $O(n^2)$ time and $O(n^2)$ space complexity. Sibson's variant of SLINK[9] is the best-known variant that has $O(n^2)$ time and $O(n)$ space complexity. This algorithm merges a pair of clusters with minimum inter-cluster distance at each iteration. It has reduced space complexity by storing only a single row of the distance matrix at a time.

The early approaches of parallel hierarchical clustering are based on similarity matrix[90–92]. The first parallel hierarchical algorithm was presented in [90]. It has a time complexity of $O(n^2)$ and was based on the single instruction multiple data model (SIMD) that uses a shuffle exchange network to access similarity matrix and input data. The next approach was presented in [91], which uses reconfigurable optical buses (AROB) architecture. The limitations of the above two approaches is that they are designed for specialized parallel architectures. In 2005, an MPI-based approach was presented[92]. In this approach, the similarity matrix along with the data points is distributed across multiple nodes and then synchronized at each merging step. The clustering quality is dependent on the chosen input parameter threshold. The above similarity matrix based approaches incur a high communication cost for iteratively updating the similarity matrix. This limits their performance and scalability, and renders them unfit for processing large datasets. Note that the above approaches use random distribution, and the use of spatial partitioning has no effect on their performance.

An approximate parallel SLINK algorithm was presented by Johnson and Kargupta[93] for distributed memory systems. In this algorithm, data communication is lowered by using lower and upper bounds for the distances between any two given points. The upper and the lower bound refer to maximum possible distance between two points and the distance value stored in the lowest root of the subtree connecting the two leaves in the dendrogram, respectively. Local dendrograms using these bounds are merged to get the global dendrogram.

Among more recent approaches, SLINK algorithm has been viewed as a Minimum Spanning Tree (MST) problem[34]. This is because SLINK has "the same agglomerative nearest neighbor property" by virtue of which, if a cluster has its nearest neighbor as cluster i or cluster j, the merged agglomerative cluster $i+j$ will be the nearest neighbor of that cluster[37]. Many researchers have attempted to solve parallel SLINK as a parallel MST construction problem. [94] presents two parallel solutions for SLINK specific to the shared memory PRAM model and distributed memory parallel machine with butterfly architecture. Then, an MPI-based distributed memory parallel clustering algorithm, known as CLUMP was proposed[37]. They consider whole data as a graph, which is partitioned randomly into smaller sub-graphs composed of complete bipartite graphs. Then an MST is computed for each sub-graph. They also compute MSTs for self-edges and cross-edges by distributing the load among multiple nodes. These local MSTs are then merged to get the final MST. The basic idea is to minimize the communication cost at the expense of redundant computations. Another approach was presented in [59], which gave an efficient parallel hierarchical algorithm using parallel Euclidean Minimum Spanning Tree (EMST). This approach assumes that the data points are uniformly distributed, allowing for partitioning of the data space into uniform grids.

The next parallel MPI-based algorithm was PINK[34], which is similar to CLUMP. They also minimize the communication cost by decomposing the problem into sub-problems that removes redundant computations at the same time. This approach partitions the data into k equal partitions randomly and assign each $\binom{k}{2}$ possible combinations to various nodes for cross-edge and self-edge computations. At each node, a local clustering is performed where an MST is computed locally. Then these MSTs are merged into a global MST resulting in the final dendrogram. The major drawback of this technique is that, for increasing k (and so p), redundant data will keep on increasing exponentially which may affect the scalability of the algorithm. A similar algorithm known as SHRINK[36] was also proposed for shared memory systems. Both the above approaches use the Union-Find data structure for merging clusters at each iteration.

The Sibson's SLINK algorithm does not take into account spatial locality of data. And thus all its parallelizations (discussed above) use random data distribution. The data partitioning scheme has no affect on the performance of the algorithm. An efficient HAC algorithm known as Partially Overlapping Partitioning (POP) has been proposed which exploits spatial

locality[95]. The pPOP algorithm[60] is a parallel implementation of the POP algorithm for shared memory architectures. pPOP is an HAC algorithm that has two phases. Data is partitioned into p overlapping cells, in the first phase. In each iteration, the closest pair in each cell is found to obtain the overall closest pair. If the overall closest pair is less than an overlap threshold, δ, those pairs are merged. If the distance of the closest pair exceeds δ, phase 2 performs hierarchical clustering. The authors also proposed the design of a data structure called a hyper-plane clustering tree (hpc-tree) for indexing higher-dimensional data.

The most recent parallel version of SLINK is *dGrid*SLINK[35]. It is a parallel version of the GridSLINK algorithm (proposed by the same authors). *Grid*SLINK exploits spatial locality of data while processing, and reduces the number of distance calculations while producing the exact dendrogram as that of classical SLINK. *Grid*SLINK has been parallelized for distributed memory (*dGrid*SLINK), shared memory (*sGrid*SLINK) and hybrid architectures (*hGrid*SLINK). *dGrid*SLINK ensures load-balancing by spatially distributing equal amounts of data to multiple nodes using a spatial distribution and adaptive gridding (referring to Subsection 3.5 for details). After data distribution, at each node GridSLINK is executed leading to local MSTs. Local computations in GridSLINK are more optimal than in PINK as this exploits spatial locality attained by the grid. Then the local MSTs are merged into a global MST in a tree-parallel way to get the final dendrogram.

Apart from these a few MapReduce and Spark-based implementations have also been proposed[96, 97]. They employ random distribution. Recently, another Spark-based parallel hierarchical clustering algorithm for complete linkage, known as PACk has been proposed in [98]. PACk uses distance-aware partitioning to partition the data to include a set of nearest neighbours in the same partition for each item. It thereby allows more merges to happen within each partition. This algorithm also uses a distance-aware merging algorithm that computes distance bounds to safely merge as many mutual nearest neighbours as possible inside a partition. The experimental results show its better performance when compared with the state-of-the-art.

2.4 Parallel Subspace Clustering

Subspace clustering algorithms are specifically designed for processing high-dimensional datasets. It is possible that data points might have been drawn from multiple subspaces and the membership of points to those subspaces is unknown. Another problem associated with the processing of high-dimensional data is the curse of dimensionality. The conventional similarity measures become unfit for processing such high-dimensional data. Subspace clustering algorithms are a solution to the above problems. They cluster data into multiple subspaces and find a low-dimensional subspace fitting each cluster. There are two kinds of subspace clustering algorithms, top-down and bottom-up. The top-down subspace clustering algorithms produce highly disjoint clusters using partitioning-based clustering approaches. A few top-down subspace algorithms include PROCLUS[13], ORCLUS[14], FINDIT[15], δ-Clusters[99], COSA[100], and LAC[101]. Bottom-up subspace clustering is similar to finding frequent itemsets using the apriori principle. Clusters are first found for every single dimension, and then dimensions are added to find clusters in higher dimensions in the same way as that of apriori. Dimensions are added until cluster quality is preserved. The anti-monotonic property is used to prune away infrequent or irrelevant subspaces. Commonly used grid-based bottom-up subspace clustering algorithms include CLIQUE[10], MAFIA[11], ENCLUS[12], SCHISM[102], and CBF[103]. Also, there are a few density-based bottom-up subspace clustering algorithms that include SUBCLUE[104], FIRES[105], DUSC[106], INSCY[107], and SUBSCALE[108].

Literature reveals very few approaches to parallel subspace clustering on distributed memory architectures. The first such approach is the parallelization of MAFIA known as PMAFIA[61]. This algorithm uses random distribution, and then performs local computations on each node, whose results are then merged into a global output. A GPU-based parallelization of MAFIA has also been presented in [32].

More recently a parallel framework[33] has been presented for grid-based bottom-up subspace clustering algorithms like CLIQUE, MAFIA, ENCLUS, SCHISM, and CBF. This framework has five major steps: 1) gridding, 2) finding dense units, 3) candidate unit/subspace generation for the next iteration, 4) steps 2 and 3 are repeated until no dense units are found, and 5) cluster extraction. These steps are common to the above bottom-up subspace clustering algorithms. The parallel framework first distributes the data randomly over the computing nodes, and then every node executes steps 1–3 iteratively. At each it-

eration, a local trie is generated at every node, which is communicated to the master to form a global trie for dense unit identification. This is repeated until the algorithm converges. Finally, the clusters are extracted at the master node from the aggregates received.

The above approaches use random data distribution and do not rely on spatial locality by their design. Hence, we do not consider them for experimentation in Section 4.

Apart from the above, the top-down subspace clustering algorithm LAC has been parallelized for shared memory architecture, known as PLAC[31]. A spark-based parallelization of the SUBCLUE algorithm, known as CLUS, is also presented in [109]. More recently, a grid-based parallel subspace clustering algorithm known as PSCEG[110] has also been presented for Spark. A MapReduce-based parallel subspace clustering is presented in [111]. All of these approaches employ random data distribution.

3 Data Distribution Strategies

We now describe the data distribution strategies, including existing strategies (Random, KD-Split, Quad-Split) and proposed strategies (PD-Split, CD-Split, and Pbased-Split), and their impact on parallel clustering algorithms. Most of the illustrated existing/proposed distribution strategies are only slightly different in their approach to partitioning. However, they cause a considerable effect on the overall performance of the parallel clustering algorithms, which can be evidently seen from experiments presented in Section 4.

Let N and d be the size and dimensionality of the dataset, respectively, and p be the total number of computing nodes or processing elements in the cluster.

3.1 Random Partitioning

In random partitioning, data points are randomly distributed to the computing nodes of the cluster. In practice, the first chunk of N/p data points are assigned to the first computing node, the next chunk to the second node, and so on. Data load balancing is maintained in the distribution to achieve better performance, i.e., each computing node gets an equal number of data points. A few examples of the algorithms that use random partitioning include [21, 34, 38]. In general, random partitioning is suitable for algorithms that are embarrassingly parallel (like k-means). When used for density-based or hierarchical clustering algorithms, we are not making the best use of the algorithm's inherent spatial pattern of execution, which degrades their performance. Instead, it is better to use one of the spatial partitioning schemes (explained below), which captures such inherent spatial patterns.

3.2 *kd*-Tree Based Split

kd-tree based split or KD-Split is the most commonly used spatial partitioning technique for data distribution[24, 28, 33, 39, 40]. This technique recursively divides data amongst the computing nodes based on an axis-aligned split (see Fig.2). For every division, the axis that has the largest spread is chosen and the split is performed based on the median for perfect data load balancing. The recursive division continues until the total number of partitions is equal to the total number of computing nodes. Since load balancing is maintained at each split, each computing node gets an equal number of data points. Fig.2 illustrates KD-Split for p=8. It shows stage-by-stage splitting, where the median for each split is chosen across the dimension that has the largest spread.

3.3 Quad-Tree Based Split

Quad-tree based split or Quad-Split is a spatial partitioning technique that employs the splitting method used by quad-trees (used in [48]). For each split, the data is split in 2^d partitions of equal sizes as shown in Fig.3. The partitioning continues recursively until the number of partitions is equal to the number of computing nodes. This kind of data distribution produces spatial disjointed partitions but does not guarantee data load balancing, especially when applied over skewed datasets. Hence, quad-tree is not commonly used for data distribution. *kd*-tree based distribution is better than that of quad-tree as it achieves good load balancing.

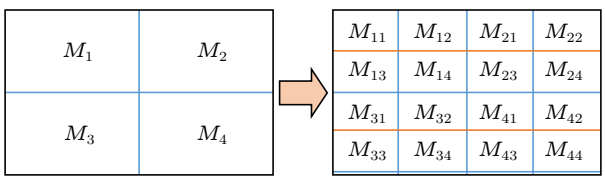


Fig.3. Quad-tree based data partitioning or Quad-Split[48].

3.4 Parameterized Dimensional Split

Parameterized Dimensional Split or PD-Split is the first proposed partitioning scheme (Fig.4). This is specifically designed for parallel density-based clustering algorithms[24, 28, 39]. This partitioning scheme strives to minimize the communication overhead required during their execution. A typical parallel density-based clustering algorithm has the following execution layout.

• In step 1, data is distributed to the computing nodes using a spatial partitioning scheme (typically KD-Split).

• In step 2, every computing node requests for data points from other nodes, which are lying within ϵ-extended boundaries of a local node. ϵ is a user-defined density parameter. This is depicted in Fig.5 for kd-tree partitioning, where node M_{212} requests data points from nodes M_{121}, M_{122}, M_{211}, and M_{222}. These data points are required for computing exact ϵ-neighborhoods for the points lying near the boundaries of the local computing node. After every computing node fetches data from the other nodes, local computations are performed where DBSCAN is performed on the local data with the help of additionally retrieved data.

• In step 3, local clusterings are merged into a global clustering output.

Exploiting this execution layout, we try to minimize the communication overhead that occurs during step 2 of the algorithm by modifying the kd-tree partitioning scheme. Instead of computing a new axis for each division, we let the division happen across the initially chosen dimension until a threshold is reached. The threshold is on the width of the cell along the chosen splitting axis. If a division is causing a cell's width to be a threshold $\leqslant 2\epsilon$, we then choose the next dimension that has the largest spread for the division. Fig.4 illustrates this. In the first and the second splits, the division has occurred only along the x-axis. However, in the third recursive split, partition M_{11} was divided along y-axis. This is because the width of one of the cells resultant of splitting this partition along the x-axis, was becoming lesser than 2ϵ. Therefore, the axis for the division was changed.

Figs.5 and 6 illustrate the the ϵ-extended strips (also known as halo region) for partition M_{212} for KD-Split and PD-Split, respectively. It is clear from the figure that the halo region spawns four partitions in the case of KD-Split and only two partitions in the case of PD-Split. When the dimensionality of the dataset increases, the number of nodes overlapping can even be more in KD-Split partitioning as the axis for split keeps changing for every division. Therefore, PD-Split reduces the data required to be communicated in steps 2 and 3 of a parallel density-based clustering algorithm as it reduces the number of nodes to be approached for acquiring extra data points.

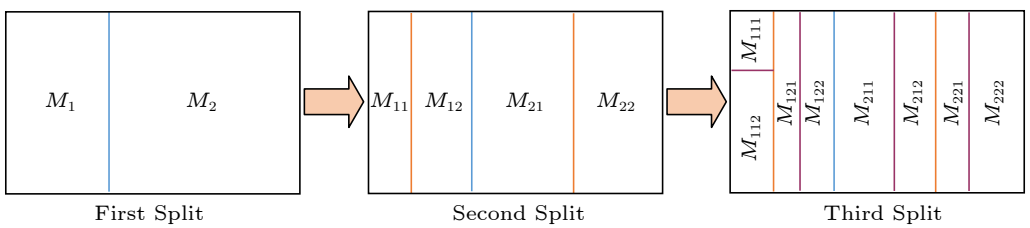


Fig.4. PD-Split data partitioning.

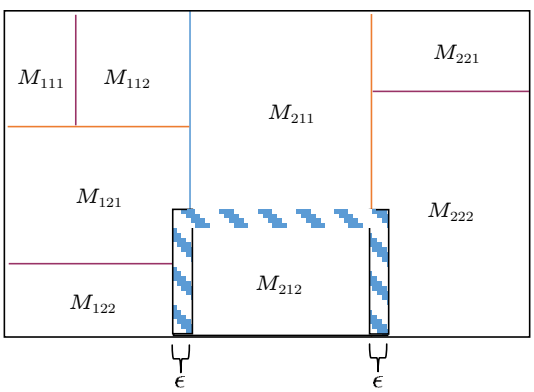


Fig.5. ϵ-extended regions for computing node M_{212} in case of KD-Split.

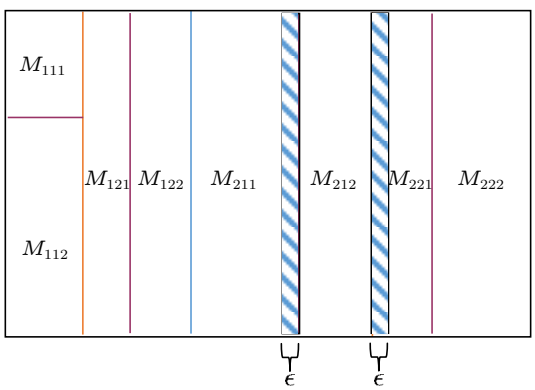


Fig.6. ϵ-extended regions for computing node M_{212} in case of PD-Split.

Note that the threshold on the width of each resultant partition has been chosen to be 2ϵ. This is because if the width of the partition becomes less than ϵ, the ϵ-extended strip might spawn to multiple partitions across the same axis. For example, in Fig.6, if the width of partition M_{121} is lesser than ϵ, the ϵ-extended strip of M_{112} can spawn to machine M_{122} as well, which means that we are including all of M_{121} and some portion of M_{122} as well. This becomes a huge communication cost. Therefore, we restrict the width of each cell to be larger than 2ϵ and whenever a split can cause the width to go less than this value, we change our splitting axis.

3.5 Cell-Based Dimensional Split

Cell-based Dimensional Split or CD-Split is our second proposed partitioning scheme. This scheme has been specifically designed for grid-based parallel SLINK (dGridSLINK) algorithm and has been introduced in [35]. dGridSLINK is the only MPI-based parallel variant of SLINK that uses the spatial distribution of data points.

The CD-Split partitioning is performed using gridding and median-based split. It is similar to PD-Split, and additionally uses gridding. Initially, a uniform virtual grid is overlaid on the entire data space, with an initially chosen cell size: $CellSizeinit/r$. *CellSizeinit* is the cell size parameter of the GridSLINK algorithm and r (> 1) is a constant. For example, *CellSizeinit* is calculated using the formula $\sqrt[d]{RegionSize \times \tau / N}$, where *RegionSize* is the volume of the data-space occupied by the points in the dataset, N is the size of the dataset and τ is a user-defined threshold on the maximum number of points we wish to keep in a cell. After gridding, we recursively split the data space into equal partitions by first splitting along one dimension, similar to PD-Split. Each split is a median-based split. However every time, the splitting axis is aligned with the nearest cell boundary as illustrated in Fig.7. The change in the dimension for splitting in case of CD-Split, however, is triggered by the cell size threshold, instead of ϵ-threshold like in PD-Split. The dimension for splitting is changed when the partition width can fit in only one cell across the current dimension. The total number of dimensions across which splitting is performed usually remains small, similar to PD-Split.

As mentioned before, CD-Split is designed for the dGridSLINK algorithm, but can be used for many grid based parallel aglorithms. The dGridSLINK algorithm internally performs local gridding in the local computations phase and performs the SLINK clustering using the internal grid info. The local computations phase also makes use of the global gridding and partitioning performed (with axis alignment to the grid) during the data distribution phase. This makes the local computations faster as the partitioning is in accordance with gridding. One can use KD-Split or Pbased-Split (to be explained next) instead of CD-Split as well. However, the algorithm is expected to run slower for them as they do not do the split-axis alignment. This is substantiated by experiments presented in Subsection 4.3 (for more details, refer to [35]).

3.6 Projection-Based Split

Projection-based split, or Pbased Split, is our third proposed partitioning scheme. This is a generic strategy and is not designed for any specific algorithm. In this partitioning, the axis with the largest spread is chosen, and the data is recursively divided into partitions based on the median. Each division is done along the same axis. The recursive division continues until each partition (or cell) contains a total number of points smaller than α, where α is the parameter threshold. Fig.8 illustrates this split.

After the division is complete, all the cells formed are projected onto the axis chosen for splitting. The cells formed in Fig.8(a) are projected over x-axis. This results in an ordering (increasing order) among the cells over their x-coordinate values. Following this increasing order, contiguous cells are packed together

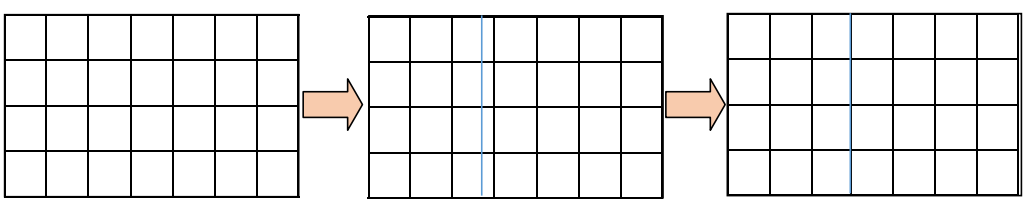


Fig.7. Sample division in CD-Split.

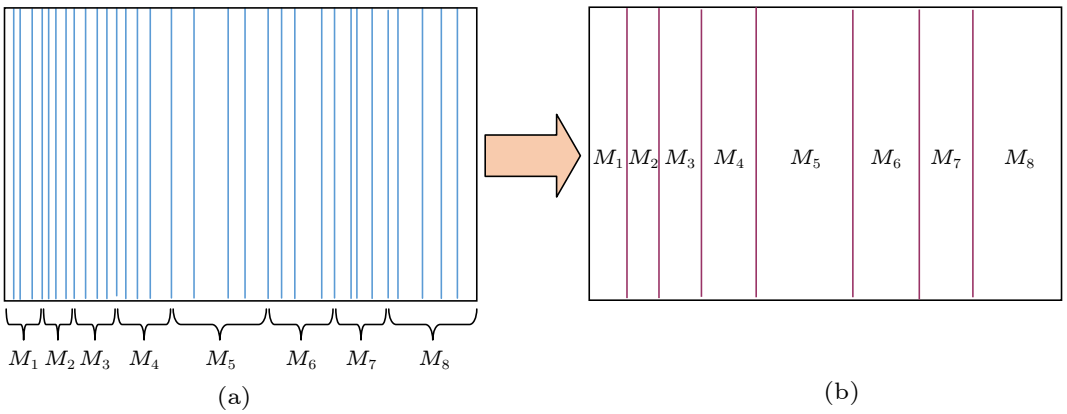


Fig.8. Pbased-Split. (a) Initial partitioning into cells. (b) Final partitioning after packing.

into non-overlapping groups (or partitions) in such a way that each group does not contain more than $\lceil N/p \rceil$ points (Fig.8(b)). This scheme results in load balancing, which is very close to perfect load balancing. We can observe that the smaller the value of α is, the better the load balancing would be.

3.7 Analyzing PD-Split and Pbased-Split

We observe that PD-Split and Pbased-Split follow similar principles while partitioning, except that the splitting axis never changes in Pbased-Split. Both of these distribution strategies are intended to reduce the number of nodes communicated during steps 2 and 3 of the parallel clustering algorithm, and thus reduce the communication overhead when compared with KD-Split. This is achieved by minimizing the number of computing nodes that overlap with the ϵ-extended boundaries of data present in any given computing node. The benefit is more tangible while processing high-dimensional datasets as they significantly reduce the number of overlapping computing nodes when compared with KD-Split. This is substantiated by the results presented in Figs.9 and 10.

However, in certain circumstances, both these strategies could increase the total number of data points transferred, despite accessing less computing nodes. This could happen when the perimeter of the ϵ-extended region increases, which in turn can arise with an increase in the value of ϵ. This problem could be more severe in the case of Pbased-Split, as the dividing axis is never changed. All of this is substantiated by the results presented in Fig.11.

Also, note that PD-Split and Pbased-Split are expected to give better performance than KD-Split in case of heterogeneous architectures and parallel setups with low-bandwidth interconnect, as they significantly reduce the number of computing nodes to be communicated.

3.8 Data Distribution Strategies for Very Large Datasets

The distribution strategies described in Sections 3.1–3.7, load the entire data into main memory for computing the partitioning splits. However, while processing very large datasets (billions of floating points), the memory associated with the node performing the partitioning may not be sufficient to load the entire dataset. This makes those methods unfit for distributing very large datasets. To handle such scenarios one can use sampling-based techniques for data distribution. One such technique has been proposed in [26] and [41]. We name this technique as A-KD-Split:

1) All the data points are randomly distributed to all the computing nodes in the cluster.

2) A small fraction of data points are randomly selected from each computing node and are broadcasted to all other nodes in the cluster.

3) Every computing node has the same data sample now. Each computing node now computes the first median for splitting over that global sample. The medians computed across all the nodes would be exactly the same (say (x_m, y_m) for a 2-D dataset).

4) Every computing node now partitions the data it has into two partitions, with respect to the median (x_m, y_m). The axis that has the maximum spread is chosen for splitting (say x-axis for illustration purposes). One partition would contain data objects that lie on the left side of the median (objects whose x-coordinate value is less than x_m). The other partition contains data objects that lie on the right side of the median (objects whose x-coordinate value is greater than or equal to x_m).

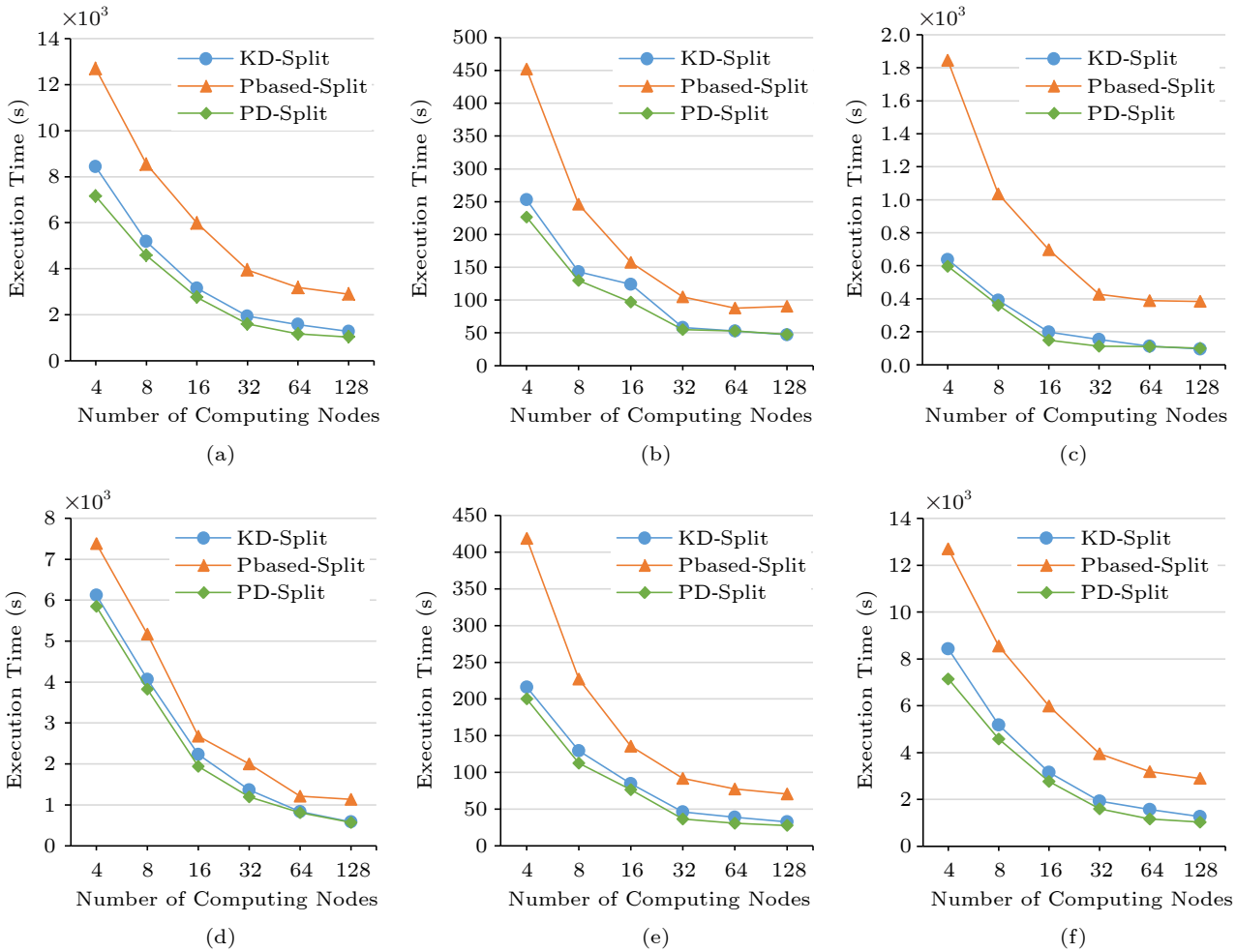


Fig.9. Performance of the GridDBSCAN-D algorithm for various data distribution strategies with various number of computing nodes of the cluster, over the (a) 3DSRN, (b) Bertone8M3D, (c) MPAHALO2.8M9D, (d) MPAGD100M3D, (e) SBus8M2D, and (f) MPAGD7M30D datasets.

5) Then in a pair of 2, computing nodes exchange their left and right partitions such that one computing node gets the entire left half and the other gets the entire right half, with respect to the global median and the axis chosen to split. This has created two global partitions amongst the computing nodes of the cluster. Half of the nodes contain data lying towards the left of the median and the other half of the nodes contain data lying towards the right of the median.

6) Now for all the machines that contain data of the left partition, steps 2–4 are repeated recursively. They are also recursively repeated for machines in the right partition. In this way, at the end of the partitioning procedure, data contained in each computing node would be completely disjoint.

7) Thus, this algorithm achieves disjoint partitioning in $\log p$ iterations.

Note that this partitioning scheme may not lead to perfect load balancing. However, it is experimentally observed to give reasonably good load balancing (see Subsection 4.2). Also, note that the approximate versions of PD-Split (A-PD-Split) and CD-Split (A-CD-Split), are also designed similarly. In the case of A-PD-Split, the nodes shall also have to additionally keep track of and communicate the dimension of the split. In the case of A-CD-Split, the initial virtual grid is to be calculated globally by inter-node communication, and a copy of the gridding information is to be broadcasted to each node. Then the splitting starts similarly to that of A-KD-Split, except that the dimension across with the split has to happen shall change as per the cell size. Also, at every split, the split-axis is aligned with the nearest cell boundary of the global grid. For the case of Pbased-Split, such an iterative distribution is not possible. The entire partitioning has to happen on the first taken sample and the data points are eventually distributed to their respective partitions, without any kind of iterative re-

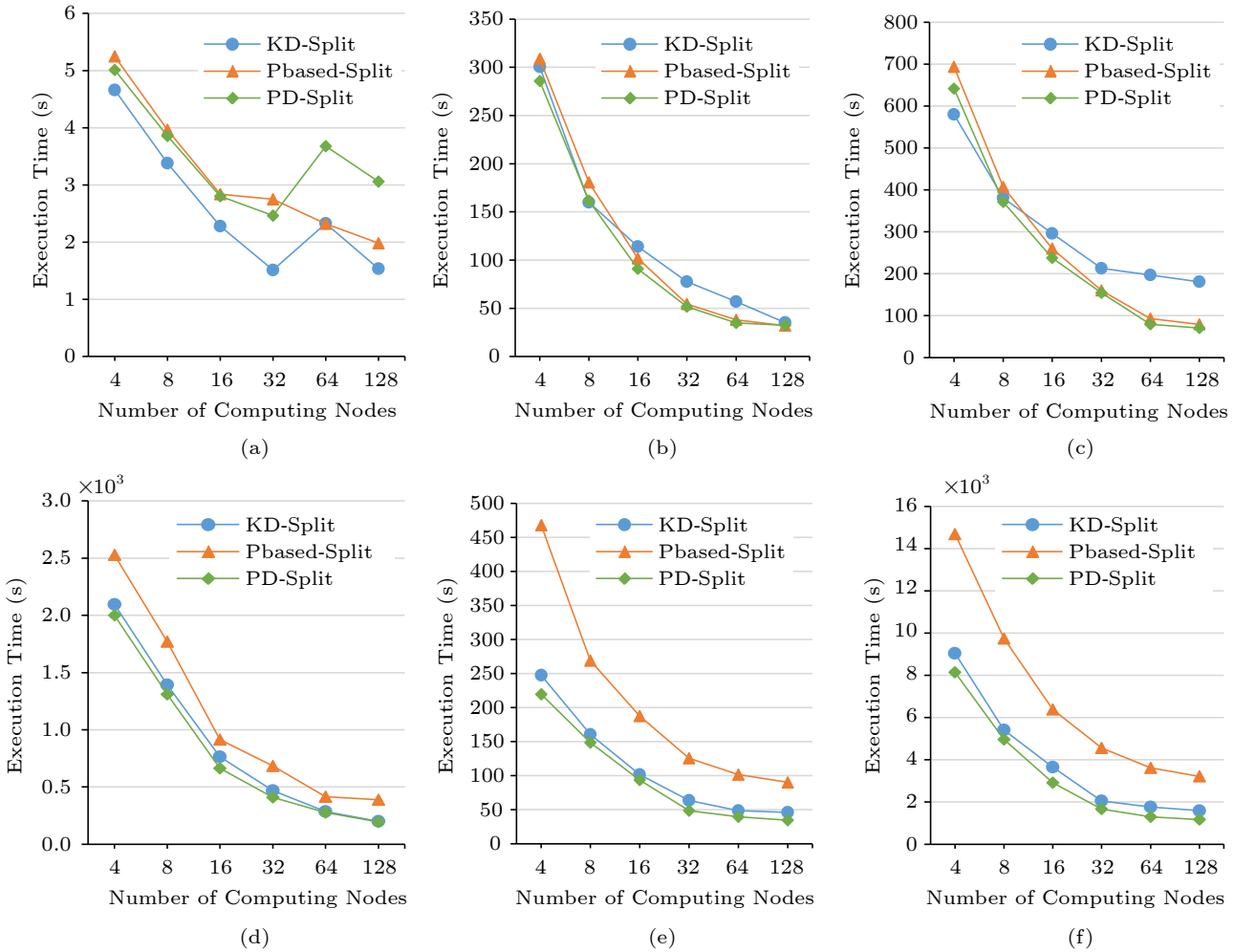


Fig.10. Performance of the PDSDBSCAN-D algorithm for various data distribution strategies with various number of computing nodes of the cluster, over the (a) 3DSRN, (b) Bertone8M3D, (c) MPAHALO2.8M9D, (d) MPAGD100M3D, (e) SBus8M2D, and (f) MPAGD7M30D datasets.

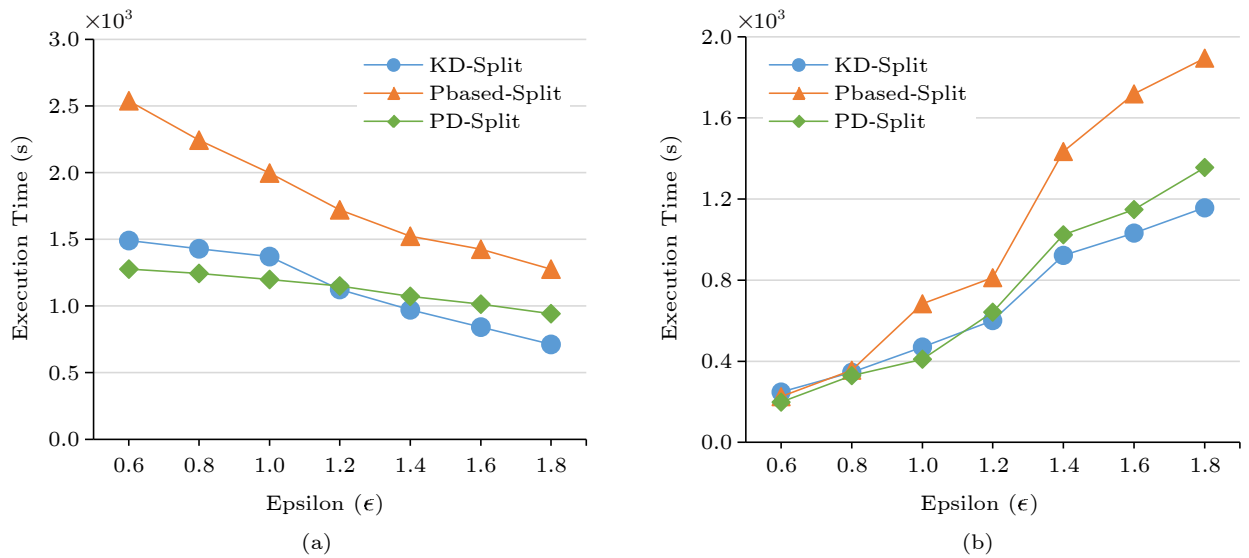


Fig.11. Performance of (a) GridDBSCAN-D and (b) PDSDBSCAN-D with variation in ϵ for various distribution strategies over 32 computing nodes for the MPAGD100M3D dataset.

finement. This may not result in good load balancing and hence we omit it for further discussion.

4 Experiments

4.1 Experimental Setup

All experiments are conducted on a cluster of 32 computing nodes, which are IBM x3250 m4 servers that are connected via gigabit ethernet. Each server has an Intel Xeon (64-bit) processor and 32 GB RAM. All implementations are in C/C++ with MPI. The details of the datasets used for experimentation are given in Table 2. The 3DSRN[112] dataset contains geographical information (latitude, longitude, and altitude) of road networks in Denmark. The MPAGD, Bertone, MPAHALO, and FOF datasets are taken from Millennium data repository[113] that contains astronomical data of galaxies. The SBus dataset① contains GPS traces of buses in Shanghai. We synthetically generate the SR10M3D, SU10M3D, and SN10M3D datasets, whose description is given in Subsection 4.4.

The execution time for each experiment has been measured using `MPI_Wtime()` of the MPI library. The default value of α is chosen to be 1 000 for Pbased-Split. This is small enough for handling data of millions scale to achieve good load balancing. For approximate distributions based on sampling, we choose 10% points of the dataset as the sample. Note that for datasets of size larger than 20M, we use approximate sampling-based distributions.

We evaluate the proposed data distribution strategies in terms of 1) data load balancing achieved, and 2) performance of various parallel spatial clustering algorithms. The results are presented as follows.

4.2 Data Load Balancing Achieved

Table 3 shows the data load balancing achieved for each of the distribution strategies for the FOF57M3D dataset. Each value in the table denotes the number of data points received per computing node. As explained earlier, random partitioning, KD-Split, and PD-Split achieve perfect load balancing. Pbased-split achieves near perfect load balancing because of the packing techniques it employs as explained in Subsection 3.6. Similarly, CD-Split also achieves near perfect load balancing as the splitting boundaries get aligned with grid/cell boundaries (as explained in Subsection 3.5). Please note that a similar load balancing has been observed for the other datasets as well.

4.3 Performance of Parallel Spatial Clustering Algorithms

4.3.1 *Parallel DBSCAN*

We compare the performance of PDSDBSCAN-D[24] and GridDBSCAN-D[28] for various distribution strategies. The ϵ value for each dataset has been given in Table 2. The value of *min_pts* has been set to 5. Fig.9 and Fig.10 present the performance of GridDBSCAN-D and PDSDBSCAN-D respectively, for KD-Split, Pbased-Split, and PD-Split distributions for

Table 2. Details of Datasets Used for Experimentation and Their Experimental Parameters

Dataset	Size	Number of Dimensions	Parameter for PDSDBSCAN-D and GridDBSCAN-D
3DSRN[112]	434K	3	ϵ=0.01, *min_pts*=5
Bertone8M3D[113]	8M	3	ϵ=2, *min_pts*=5
MPAHALO2.8M9D[113]	2.8M	9	ϵ=30, *min_pts*=5
MPAGD100M3D[113]	100M	3	ϵ=1, *min_pts*=5
SBus8M2D*	8M	2	ϵ=1, *min_pts*=5
MPAGD7M30D[113]	7M	30	ϵ=6, *min_pts*=5
MPAGD2M30D[113]	2M	30	ϵ=6, *min_pts*=5
MPAGD16M3D[113]	16M	3	ϵ=2, *min_pts*=5
FOF57M3D[113]	57M	3	ϵ=3, *min_pts*=5
SR10M3D	10M	3	ϵ=0.01, *min_pts*=5
SU10M3D	10M	3	ϵ=0.01, *min_pts*=5
SN10M3D	10M	3	ϵ=0.01, *min_pts*=5

①SUVN trace data. http://wirelesslab.sjtu.edu.cn/, Sept. 2015.

Table 3. Number of Data Points Received by Each Computing Node for Various Data Distribution Strategies with Various Number of Computing Nodes (p), for Dataset FOF57M3D

Distribution Strategy	p=16	p=32
Random	3 561 887	1 780 944
KD-Split	3 561 887	1 780 944
PD-Split	3 561 887	1 780 944
Pbased-Split	3 541 062 to 3 571 329	1 721 712 to 1 813 961
CD-Split	3 498 032 to 3 638 541	1 597 254 to 1 862 171
A-KD-Split	3 397 251 to 3 795 134	1 584 754 to 1 922 658
A-PD-Split	3 344 652 to 3 786 249	1 571 113 to 1 911 904
A-CD-Split	3 285 412 to 3 799 763	1 523 624 to 1 924 521

various datasets executed over an increasing number of computing nodes. The results show that PD-Split and KD-Split are competitive in execution performance. We can observe that for a lesser number of computing nodes, PD-Split is better than KD-Split. However, with the increase in the number of computing nodes, both of them give almost the same performance. For high-dimensional datasets (MPAGD7M-30D and MPAHALO2.8M9D), PD-Split works much better than KD-Split, even at a higher number of computing nodes. This is because of reduced communication overhead in steps 2 and 3 of both the algorithms, as explained in Subsection 3.4. This is also substantiated by the split-up time of various steps of the algorithms presented in Table 4 and Table 5. PD-Split improves overall execution time as well as the execution time of each step of the algorithm. The results of the 3DSRN dataset are erratic at a higher number of computing nodes, because of insufficient data to be processed for such a large number of processors.

Next, we conduct an experiment to measure the performance of both the parallel algorithms with variation in ϵ value. Figs.11(a) and 11(b) present the results, which show that PD-Split works better for lower values of ϵ. Whereas, KD-Split is found to dominate for higher values of ϵ. This is because, at higher values of ϵ, the communication cost of Pbased-Split and PD-Split becomes higher as explained in Subsection 3.7.

Table 4. Split-up of Execution Time (s) of Various Steps of GridDBSCAN-D for Dataset MPAGD100M3D

Step	KD-Split	Pbased-Split	PD-Split
Data distribution + retrieval of extra points	26.58	37.23	19.33
Local computations	1 174.91	1 673.60	1 023.45
Merging	167.92	287.34	149.34
Total time	1 369.43	1 998.23	1 192.12

Table 5. Split-up of Execution Time (s) of Various Steps of PDSDBSCAN-D for Dataset MPAGD100M3D

Step	KD-Split	Pbased-Split	PD-Split
Data distribution + retrieval of extra points	26.58	37.23	19.33
Local computations	376.23	508.72	305.53
Merging	92.51	138.01	79.23
Total time	468.72	683.95	404.09

4.3.2 *Parallel SNN*

In this subsection, we evaluate the performance of the dR-SNN algorithm[40]. The values of the parameters chosen for experimentation are: k=30, ϵ=12, and min_pts=15, for all datasets. Note that ϵ of SNN is different from that of DBSCAN. It is a threshold on the number of data points in the case of SNN and a threshold on the distance in the case of DBSCAN. Fig.12 presents the execution time of dR-SNN for KD-Split and Pbased-Split distributions for various datasets executed over an increasing number of computing nodes. The results clearly show that KD-Split has always been better than Pbased-Split. This is because: 1) kNN queries work better in the case of globular regions, and 2) KD-Split produces more cubical-shaped regions in comparison with Pbased-Split. Therefore, the merging step in dR-SNN requires more communication in the case of Pbased-Split. This is because, in the case of Pbased-Split, the number of points that participate in the merging step is large. Both these arguments are substantiated by the split-up values presented in Table 6, which clearly shows the difference in the local computations step, as well as the merging step.

4.3.3 *Parallel SLINK*

Fig.13 presents the performance of dGridSLINK[35] (with parameter $\tau = 300$) for KD-Split, Pbased-Split, and CD-Split distributions for various datasets executed over an increasing number of computing nodes. The value of τ, which dictates the initial cell size has been set to 300 as per the recommendations given in [35]. The results clearly show that CD-Split has always been better. This is because of the time reduction in the global merging step, which was made possible by adjusting the partition boundaries to align with grid boundaries. The split-up of the execution time of various algorithm steps is presented in Table 7 for the MPAGD16M3D dataset. The results clearly show that CD-Split takes more time to distribute da-

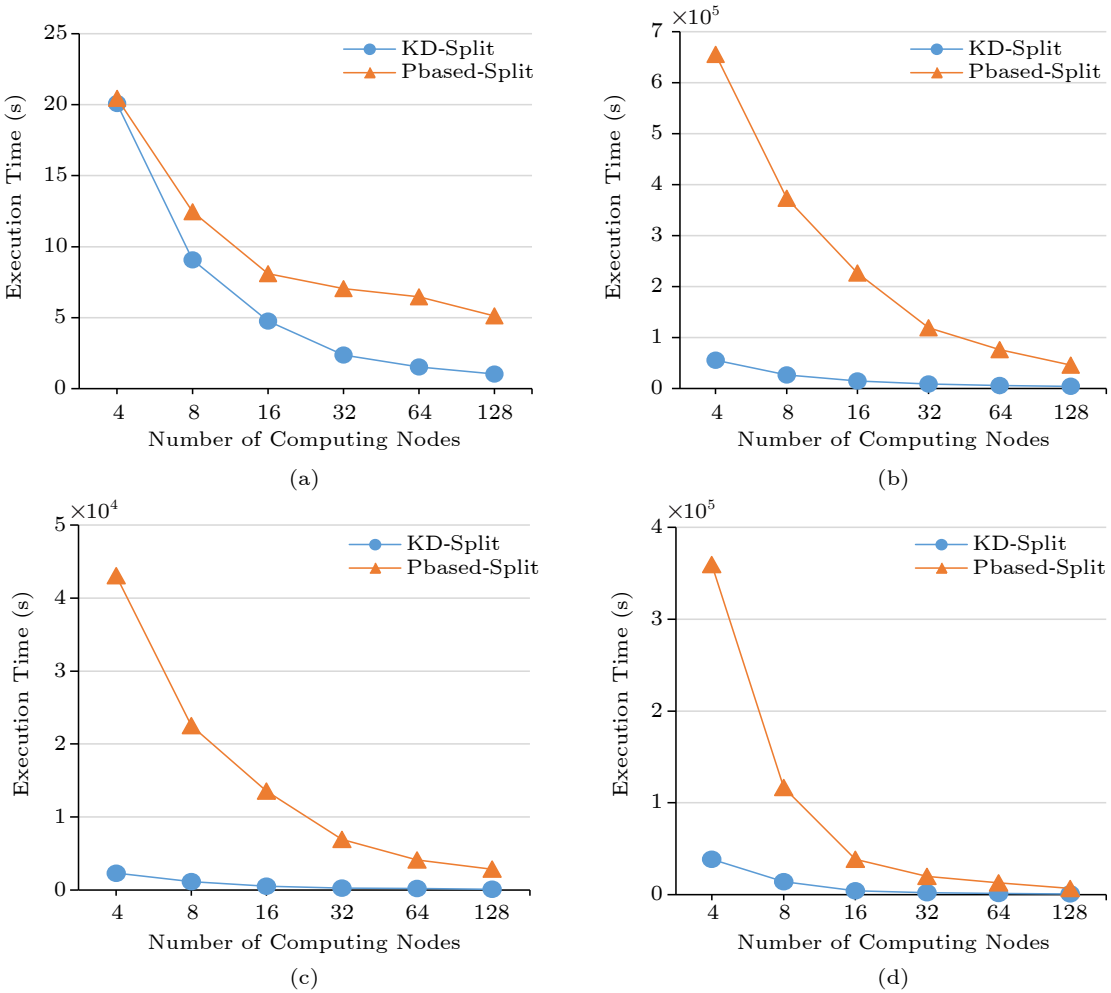


Fig.12. Performance of parallel dR-SNN for various data distribution strategies with variation in the number of computing nodes of the cluster for the (a) 3DSRN, (b) MPAGD7M30D, (c) MPAGD16M3D, and (d) FOF57M3D datasets.

Table 6. Execution Time (s) for Various Steps of Algorithm dR-SNN for Dataset MPAGD16M3S

Step	KD-Split	Pbased-Split
Data distribution	49.81	63.84
Local computations	171.86	5 192.13
Merging	27.40	1 661.48
Total time	249.07	6 917.45

ta, due to the extra load of aligning splits with grid/cell boundaries. However, the time saved in local computations and merging steps compensates for it. The merging time is especially very low for CD-Split for the above reasons. On the whole, CD-Split is better.

4.4 Experiments on Synthetic Data

In this subsection, we analyze the performance of parallel clustering algorithms on synthetic datasets of various characteristics. We specifically use three datasets: SR10M3D, SU10M3D, and SN10M3D. All three datasets contain 10M data points with three numerical dimensions. All coordinate values of data points lie in the range [–1, 1]. SR10M3D has its data objects randomly distributed, whereas SU10M3D contains data objects that are uniformly distributed across the space. The SN10M3D dataset contains data objects in a normal distribution with μ=(0, 0, 0) and σ=0.2. The parameter values of these datasets chosen for experimentation are depicted in Table 2.

We execute all four parallel clustering algorithms (PDS-DSBCAN-D, Grid-DBSCAN-D, dR-SNN, and dGridSLINK) for 16 and 32 nodes over various data distributions, and measure their running time. The results presented in Table 8 show that the algorithms run faster for the random and uniform distributions, with uniform distribution being the fastest. However, for the normal distribution dataset, the execution time is very high for all algorithms and all data distri-

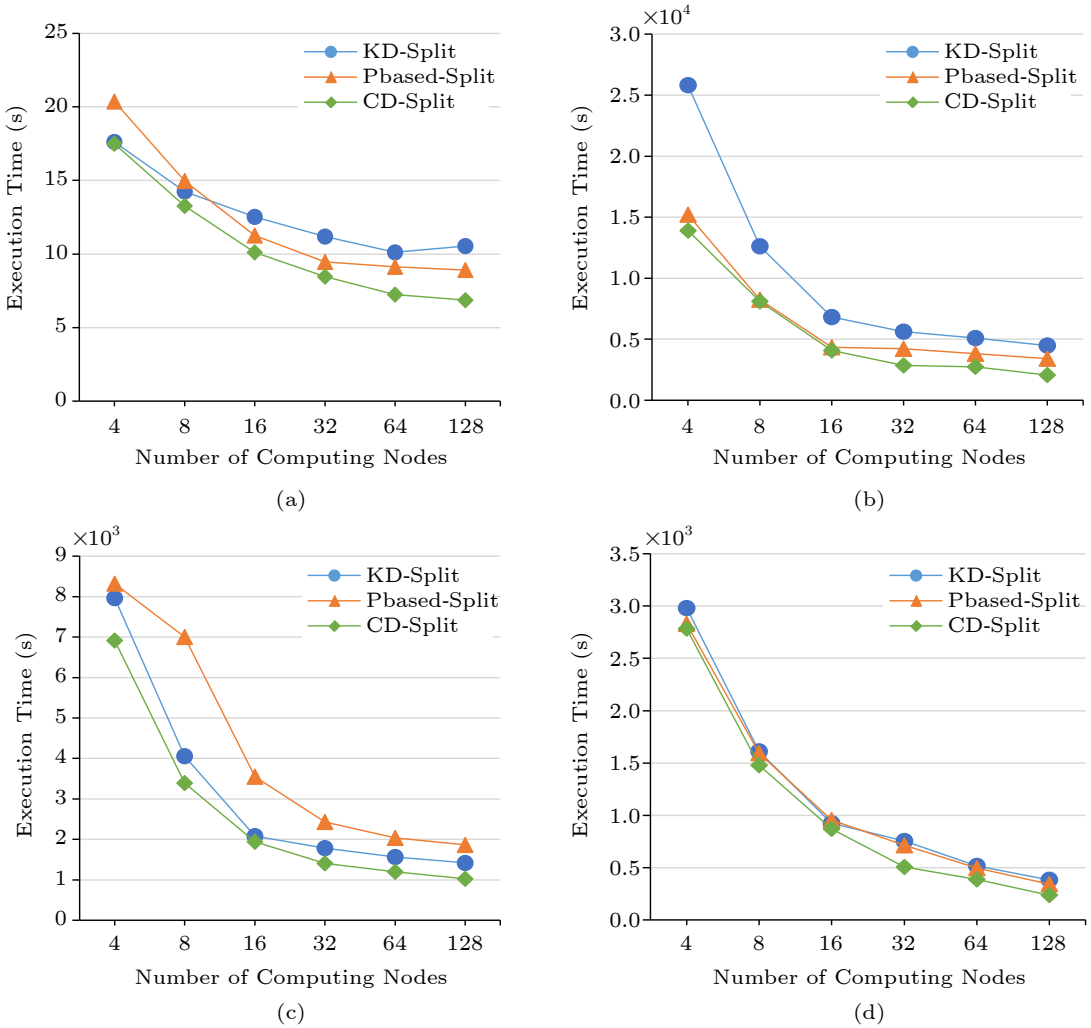


Fig.13. Performance of the dGridSLINK algorithm ($\tau = 300$) for various data distributions with variation in the number of computing nodes of the cluster for (a) 3DSRN; (b) MPAGD2M30D; (c) MPAGD16M3D; (d) FOF57M3D datasets.

Table 7. Execution Time (s) for Various Steps of dGridSLINK Algorithm for MPAGD16M3D Dataset

Step	KD-Split	Pbased-Split	CD-Split
Data distribution	49.81	63.84	59.23
Local computations	1 459.09	1 961.98	1 250.35
Merging	273.23	401.34	99.30
Total time	1 782.13	2 427.17	1 408.88

butions. This is because, the density of data objects in each node varies a lot, which leads to very high local computations time for nodes containing high-density data objects and low local computation time for nodes containing data with lower density. In the local computation step of these algorithms, local spatial queries (neighbourhood and nearest neighbour queries) are used. For the same, they employ indexing structures (such as R-tree, *kd*-tree, and Grid-R-tree). These structures have a larger overlap amongst its nodes, when the data is very dense, due to which their query performance suffers. And that is why the local computation step takes longer time for computing nodes that contain dense data objects. This variation in computation time is also depicted in Table 9, wherein the range of local computation time for uniform distribution is very small when compared with the normal distribution, indicating that some nodes (containing less dense data) finish their local computation step much quickly than a few nodes (containing highly dense data).

5 Discussion and Recommendations

Based on the above experimentation and results, we give the following recommendations regarding the usage of appropriate distribution strategies for each of the above parallel clustering algorithms.

- For parallel DBSCAN (and algorithms involv-

Table 8. Execution Time (s) of Various Parallel Clustering Algorithms for Datasets SR10M3D, SU10M3D, and SN10M3D While Using 16 and 32 Computing Nodes of the Cluster for Various Data Distributions

Algorithm	Dataset	16 Nodes				32 Nodes			
		KD-Split	PD-Split	PBased-Split	CD-Split	KD-Split	PD-Split	Pbased-Split	CD-Split
PDS-DBSCAN-D	SR10M3D	296	238	260	–	213	154	160	–
	SU10M3D	272	226	235	–	196	142	147	–
	SN10M3D	958	870	1 125	–	684	617	844	–
Grid-DBSCAN-D	SR10M3D	199	346	698	–	154	273	428	–
	SU10M3D	183	307	639	–	143	239	381	–
	SN10M3D	439	730	1 521	–	358	582	956	–
dR-SNN	SR10M3D	290	–	8 784	–	162	–	5 183	–
	SU10M3D	278	–	8 138	–	150	–	4 639	–
	SN10M3D	784	–	17 948	–	408	–	8 795	–
dGridSLINK	SR10M3D	997	–	1 303	849	718	–	925	594
	SU10M3D	914	–	1 214	816	676	–	850	569
	SN10M3D	2 784	–	3 777	2 463	1 949	–	2 833	1 847

Table 9. Variation in Computational Load Measured as Execution Time (s) of the Local Clustering Step at Each Computing Node for Various Parallel Clustering Algorithms over Datasets SU10M3D and SN10M3D for 16 Computing Nodes in the Cluster

Dataset	Algorithm	KD-Split	PD-Split	PBased-Split
SU10M3D	PDS-DBSCAN-D	218 to 248	189 to 201	187 to 204
	Grid-DBSCAN-D	117 to 139	221 to 247	502 to 513
	dR-SNN	198 to 217	–	6 100 to 6 900
SN10M3D	PDS-DBSCAN-D	310 to 874	423 to 770	584 to 1 080
	Grid-DBSCAN-D	123 to 370	374 to 670	876 to 1 420
	dR-SNN	365 to 590	–	5 738 to 14 679

ing neighborhood computations), PD-Split and KD-Split are competitive. PD-Split is more suitable for smaller values of ϵ and high-dimensional datasets. KD-Split is recommended to be used for larger values of ϵ.

• For dR -SNN and algorithms that use kNN queries, KD-Split always works better.

• For dGridSLINK, CD -Split has always been better than KD-Split and Pbased-Split.

• One can use Pbased-Split as a generic distribution scheme, free from parameters, when one wants to split across one dimension only. Pbased-Split also works well for high-dimensional data in some cases (see Fig.10(f)).

• Both PD -Split and Pbased-Split are recommended to be used for heterogeneous architectures and low-bandwidth network interconnects as they minimize the area of the halo regions.

6 Conclusions

This paper discusses an important aspect of parallel clustering algorithms, the data distribution step. To the best of our knowledge, it is a first-of-its-kind paper that gives a comprehensive review and a comparative study of the data distribution strategies used in parallel clustering algorithms, along with three new strategies namely PD-Split, CD-Split, and Pbased-Split. PD-Split has been designed for parallel density-based clustering algorithms like DBSCAN and OPTICS, CD-Split has been designed for grid-based algorithms like dGridSLINK, and Pbased-Split is a generic distribution strategy. These new strategies were experimentally shown to improve the performance of their respective algorithms when compared with the state-of-the-art methods, as illustrated in Section 4. The paper also gives a very comprehensive review of MPI-based parallel clustering algorithms with specific discussion on the data distribution strategy they use.

A hybrid design of grid and PD-Split can be developed to work more efficiently for GridDBSCAN-D. More such tailor-made distribution strategies can be developed for other classes of parallel clustering algorithms like subspace and grid-based clustering.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Tan P N, Steinbach M, Kumar V. Introduction to Data Mining. Addison-Wesley Longman Publishing Co., Inc., 2005.

[2] MacQueen J. Some methods for classification and analysis of multivariate observations. In *Proc. the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Jan. 1967, pp.281–297.

[3] Park H S, Jun C H. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 2009, 36(2): 3336–3341. DOI: 10.1016/j.eswa.2008.01.039.

[4] Steinbach M, Karypis G, Kumar V. A comparison of document clustering techniques. Technical Report, TR 00-034, University of Minnesota, 2000. https://conservancy.umn.edu/handle/11299/215421, Mar. 2024.

[5] Ester M, Kriegel H P, Sander J, Xu X W. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. the 2nd International Conference on Knowledge Discovery and Data Mining*, Aug. 1996, pp.226–231. DOI: 10.5555/3001460.3001507.

[6] Ankerst M, Breunig M M, Kriegel H P, Sander J. OPTICS: Ordering points to identify the clustering structure. In *Proc. the 1999 ACM SIGMOD International Conference on Management of Data*, Jun. 1999, pp.49–60. DOI: 10.1145/304182.304187.

[7] Jarvis R A, Patrick E. Clustering using a similarity measure based on shared near neighbors. *IEEE Trans. Computers*, 1973, C-22(11): 1025–1034. DOI: 10.1109/T-C.1973.223640.

[8] Hinneburg A, Keim D A. An efficient approach to clustering in large multimedia databases with noise. In *Proc. the 4th Int. Conf. Knowledge Discovery and Data Mining*, Aug. 1998, pp.58–65. DOI: 10.5555/3000292.3000302.

[9] Sibson R. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 1973, 16(1): 30–34. DOI: 10.1093/comjnl/16.1.30.

[10] Agrawal R, Gehrke J, Gunopulos D, Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Record*, 1998, 27(2): 94–105. DOI: 10.1145/276305.276314.

[11] Goil S, Nagesh H, Choudhary A. MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report, CPDC-TR-9906-010, Northwestern University, 1999. https://grid.cs.gsu.edu/~wkim/indesfiles/papers/mafia.pdf, Mar. 2024.

[12] Cheng C H, Fu A W, Zhang Y. Entropy-based subspace clustering for mining numerical data. In *Proc. the 5th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Aug. 1999, pp.84–93. DOI: 10.1145/312129.312199.

[13] Aggarwal C C, Wolf J L, Yu P S, Procopiuc C, Park J S. Fast algorithms for projected clustering. In *Proc. the 1999 ACM SIGMOD Int. Conf. Management of Data*, Jun. 1999, pp.61–72. DOI: 10.1145/304182.304188.

[14] Aggarwal C C, Yu P S. Finding generalized projected clusters in high dimensional spaces. In *Proc. the 2000 ACM SIGMOD Int. Conf. Management of Data*, May 2000, pp.70–81. DOI: 10.1145/342009.335383.

[15] Woo K G, Lee J H, Kim M H, Lee Y J. FINDIT: A fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 2004, 46(4): 255–271. DOI: 10.1016/j.infsof.2003.07.003.

[16] Wang W, Yang J, Muntz R R. STING: A statistical information grid approach to spatial data mining. In *Proc. the 23rd Int. Conf. Very Large Data Bases*. Aug. 1997, pp.186–195. DOI: 10.5555/645923.758369.

[17] Mukhopadhyay A, Maulik U. Unsupervised satellite image segmentation by combining SA based fuzzy clustering with support vector machine. In *Proc. the 7th Int. Conf. Advances in Pattern Recognition*, Feb. 2009, pp.381–384. DOI: 10.1109/ICAPR.2009.50.

[18] Thang T M, Kim J. The anomaly detection by using DBSCAN clustering with multiple parameters. In *Proc. the 2011 Int. Conf. Information Science and Applications*, Apr. 2011. DOI: 10.1109/ICISA.2011.5772437.

[19] Madeira S C, Oliveira A L. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Computational Biology and Bioinformatics*, 2004, 1(1): 24–45. DOI: 10.1109/TCBB.2004.2.

[20] Huo S. Detecting self-correlation of nonlinear, lognormal, time-series data via DBSCAN clustering method, using stock price data as example [Ph.D. Thesis]. Ohio State University, Columbus, 2011.

[21] Zhang J, Wu G Q, Hu X G, Li S Y, Hao S L. A parallel *k*-means clustering algorithm with MPI. In *Proc. the 4th International Symposium on Parallel Architectures, Algorithms and Programming*, Dec. 2011, pp.60–64. DOI: 10.1109/PAAP.2011.17.

[22] Kumari S, Maheshwari A, Goyal P, Goyal N. Parallel framework for efficient *k*-means clustering. In *Proc. the 8th Annual ACM India Conference*, Oct. 2015, pp.63–71. DOI: 10.1145/2835043.2835060.

[23] Song H, Lee J G, Han W S. PAMAE: Parallel *k*-medoids clustering with high accuracy and efficiency. In *Proc. the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2017, pp.1087–1096. DOI: 10.1145/3097983.3098098.

[24] Patwary M A, Palsetia D, Agrawal A, Liao W K, Manne F, Choudhary A. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In *Proc. the 2012 International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2012, Article No. 62. DOI: 10.5555/2388996.2389081.

[25] Patwary M M A, Satish N, Sundaram N, Manne F, Habib S, Dubey P. Pardicle: Parallel approximate density-based clustering. In *Proc. the 2014 Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2014, pp.560–571. DOI: 10.1109/SC.2014.51.

[26] Patwary M M A, Byna S, Satish N R *et al.* BD-CATS: Big data clustering at trillion particle scale. In *Proc. the 2015 Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2015, Article No. 6.

DOI: 10.1145/2807591.2807616.

[27] Götz M, Bodenstein C, Riedel M. HPDBSCAN: Highly parallel DBSCAN. In *Proc. the 2015 Workshop on Machine Learning in High-Performance Computing Environments*, Nov. 2015, Article No. 2. DOI: 10.1145/2834892.2834894.

[28] Kumari S, Goyal P, Sood A, Kumar D, Balasubramaniam S, Goyal N. Exact, fast and scalable parallel DBSCAN for commodity platforms. In *Proc. the 18th Int. Conf. Distributed Computing and Networking*, Jan. 2017, Article No. 14. DOI: 10.1145/3007748.3007773.

[29] Song H, Lee J G. RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning. In *Proc. the 2018 Int. Conf. Management of Data*, May 2018, pp.1173–1187. DOI: 10.1145/3183713.3196887.

[30] Sarma A, Goyal P, Kumari S, Wani A, Challa J S, Islam S, Goyal N. μDBSCAN: An exact scalable DBSCAN algorithm for big data exploiting spatial locality. In *Proc. the 2019 IEEE International Conference on Cluster Computing*, Sept. 2019. DOI: 10.1109/CLUSTER.2019.8891020.

[31] Nazerzadeh H, Ghodsi M, Sadjadian S. Parallel subspace clustering. In *Proc. the 10th Annual Conference of Computer Society of Iran*, Feb. 2005.

[32] Adinetz A, Kraus J, Meinke J, Pleiter D. GPUMAFIA: Efficient subspace clustering with MAFIA on GPUs. In *Proc. the 19th Int. Conf. Parallel Processing*, Aug. 2013, pp.838–849. DOI: 10.1007/978-3-642-40047-6_83.

[33] Goyal P, Kumari S, Singh S, Kishore V, Balasubramaniam S S, Goyal N. A parallel framework for grid-based bottom-up subspace clustering. In *Proc. the 2016 IEEE Int. Conf. Data Science and Advanced Analytics*, Oct. 2016, pp.331–340. DOI: 10.1109/DSAA.2016.42.

[34] Hendrix W, Palsetia D, Patwary M M A, Agrawal A, Liao W K, Choudhary A. A scalable algorithm for single-linkage hierarchical clustering on distributed-memory architectures. In *Proc. the 2013 IEEE Symposium on Large-Scale Data Analysis and Visualization*, Oct. 2013, pp.7–13. DOI: 10.1109/LDAV.2013.6675153.

[35] Goyal P, Kumari S, Sharma S, Kumar D, Kishore V, Balasubramaniam S, Goyal N. A fast, scalable SLINK algorithm for commodity cluster computing exploiting spatial locality. In *Proc. the 18th Int. Conf. High Performance Computing and Communications*, Dec. 2016, pp.268–275. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0047.

[36] Hendrix W, Patwary M M A, Agrawal A, Liao W K, Choudhary A. Parallel hierarchical clustering on shared memory platforms. In *Proc. the 19th International Conference on High Performance Computing*, Dec. 2012. DOI: 10.1109/HiPC.2012.6507511.

[37] Olman V, Mao F L, Wu H W, Xu Y. Parallel clustering algorithm for large data sets with applications in bioinformatics. *IEEE/ACM Trans. Computational Biology and Bioinformatics*, 2009, 6(2): 344–352. DOI: 10.1109/TCBB.2007.70272.

[38] Patwary M A, Palsetia D, Agrawal A, Liao W K, Manne F, Choudhary A. Scalable parallel OPTICS data clustering using graph algorithmic techniques. In *Proc. the 2013 International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2013, Article No. 49. DOI: 10.1145/2503210.2503255.

[39] Goyal P, Kumari S, Kumar D, Balasubramaniam S, Goyal N, Islam S, Challa J S. Parallelizing OPTICS for commodity clusters. In *Proc. the 16th International Conference on Distributed Computing and Networking*, Jan. 2015, Article No. 33. DOI: 10.1145/2684464.2684477.

[40] Kumari S, Maurya S, Goyal P, Balasubramaniam S S, Goyal N. Scalable parallel algorithms for shared nearest neighbor clustering. In *Proc. the 23rd International Conference on High Performance Computing*, Dec. 2016, pp.72–81. DOI: 10.1109/HiPC.2016.018.

[41] Challa J S, Goyal P, Nikhil S, Mangla A, Balasubramaniam S S, Goyal N. DD-Rtree: A dynamic distributed data structure for efficient data distribution among cluster nodes for spatial data mining algorithms. In *Proc. the 2016 IEEE International Conference on Big Data*, Dec. 2016, pp.27–36. DOI: 10.1109/BigData.2016.7840586.

[42] Welton B, Miller B P. Mr. Scan: A hybrid/hybrid extreme scale density based clustering algorithm. Technical Report, Northwestern University, 2015. https://www.paradyn.org/papers/Welton15MrScan.pdf, Mar. 2024.

[43] Dhillon I S, Modha D S. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, Zaki M J, Ho C T (eds.), Springer-Verlag, 2000, pp.245–260. DOI: 10.1007/3-540-46502-2_13.

[44] Zhang J, Wu G Q, Hu X G, Li S Y, Hao S L. A parallel clustering algorithm with MPI-MKmeans. *Journal of Computers*, 2013, 8(1): 10–17. DOI: 10.4304/jcp.8.1.10-17.

[45] Kumar J, Mills R T, Hoffman F M, Hargrove W W. Parallel *k*-means clustering for quantitative ecoregion delineation using large data sets. *Procedia Computer Science*, 2011, 4: 1602–1611. DOI: 10.1016/j.procs.2011.04.173.

[46] Kerdprasop K, Taokok S, Kerdprasop N. Declarative parallelized techniques for *K*-means data clustering. *International Journal of Mathematics and Computers in Simulation*, 2012, 6(5): 483–495.

[47] Balcan M F, Ehrlich S, Liang Y Y. Distributed *k*-means and *k*-median clustering on general topologies. In *Proc. the 26th International Conference on Neural Information Processing Systems*, Dec. 2013, pp.1995–2003. DOI: 10.5555/2999792.2999835.

[48] Gursoy A. Data decomposition for parallel K-means clustering. In *Proc. the 5th International Conference on Parallel Processing and Applied Mathematics*, Sept. 2003, pp.241–248. DOI: 10.1007/978-3-540-24669-5_31.

[49] Di Fatta G, Pettinger D. Dynamic load balancing in parallel KD-tree k-means. In *Proc. the 10th IEEE Int. Conf. Computer and Information Technology*, Jul. 2010, pp.2478–2485. DOI: 10.1109/CIT.2010.424.

[50] Arbelaez A, Quesada L. Parallelising the k-Medoids clustering problem using space-partitioning. In *Proc. the 6th*

International Symposium on Combinatorial Search, Jul. 2013, pp.20–28. DOI: 10.1609/socs.v4i1.18282.

[51] Li Y J, Chung S M. Parallel bisecting *k*-means with prediction clustering algorithm. *The Journal of Supercomputing*, 2007, 39(1): 19–37. DOI: 10.1007/s11227-006-0002-7.

[52] Xu X W, Jäger J, Kriegel H P. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 1999, 3(3): 263–290. DOI: 10.1023/A:1009884809343.

[53] Zhou A Y, Zhou S G, Cao J, Fan Y, Hu Y F. Approaches for scaling DBSCAN algorithm to large spatial databases. *Journal of Computer Science and Technology*, 2000, 15(6): 509–526. DOI: 10.1007/BF02948834.

[54] Arlia D, Coppola M. Experiments in parallel clustering with DBSCAN. In *Proc. the 7th International Euro-Par Conference Manchester on Parallel Processing*, Aug. 2001, pp.326–331. DOI: 10.5555/646666.699596.

[55] Coppola M, Vanneschi M. High-performance data mining with skeleton-based structured parallel programming. *Parallel Computing*, 2002, 28(5): 793–813. DOI: 10.1016/S0167-8191(02)00095-9.

[56] Brecheisen S, Kriegel H P, Pfeifle M. Parallel density-based clustering of complex objects. In *Proc. the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Apr. 2006, pp.179–188. DOI: 10.1007/11731139_22.

[57] Chen M, Gao X D, Li H F. Parallel DBSCAN with priority R-tree. In *Proc. the 2nd IEEE International Conference on Information Management and Engineering*, Apr. 2010, pp.508–511. DOI: 10.1109/ICIME.2010.5477926.

[58] Yang K Y, Gao Y J, Ma R, Chen L, Wu S, Chen G. DBSCAN-MS: Distributed density-based clustering in metric spaces. In *Proc. the 35th International Conference on Data Engineering*, Apr. 2019, pp.1346–1357. DOI: 10.1109/ICDE.2019.00122.

[59] Rajasekaran S. Efficient parallel hierarchical clustering algorithms. *IEEE Trans. Parallel and Distributed Systems*, 2005, 16(6): 497–502. DOI: 10.1109/TPDS.2005.72.

[60] Dash M, Petrutiu S, Scheuermann P. pPOP: Fast yet accurate parallel hierarchical clustering using partitioning. *Data & Knowledge Engineering*, 61(3): 563-578. DOI: 10.1016/j.datak.2006.07.004.

[61] Nagesh H S, Goil S, Choudhary A. A scalable parallel subspace clustering algorithm for massive data sets. In *Proc. the 2000 International Conference on Parallel Processing*, Aug. 2000, pp.477–484. DOI: 10.1109/ICPP.2000.876164.

[62] Bradley P S, Mangasarian O L, Street W N. Clustering via concave minimization. In *Proc. the 9th International Conference on Neural Information Processing Systems*, Dec. 1996, pp.368–374. DOI: 10.5555/2998981.2999033.

[63] Deb B, Srirama S N. Parallel K-Means clustering for gene expression data on SNOW. *International Journal of Computer Applications*, 2013, 71(24): 26–30. DOI: 10.5120/12691-9486.

[64] Torti E, Florimbi G, Castelli F, Ortega S, Fabelo H, Callicó G M, Marrero-Martin M, Leporati F. Parallel K-means clustering for brain cancer detection using hyperspectral images. *Electronics*, 2018, 7(11): 283. DOI: 10.3390/electronics7110283.

[65] Sardar T H, Ansari Z. An analysis of MapReduce efficiency in document clustering using parallel K-means algorithm. *Future Computing and Informatics Journal*, 2018, 3(2): 200–209. DOI: 10.1016/j.fcij.2018.03.003.

[66] Zhou G J. Improved optimization of canopy-Kmeans clustering algorithm based on Hadoop platform. In *Proc. the 2018 International Conference on Information Technology and Electrical Engineering*, Dec. 2018, Article No. 19. DOI: 10.1145/3148453.3306258.

[67] Megarchioti S, Mamalis B. The BigKClustering approach for document clustering using Hadoop MapReduce. In *Proc. the 22nd Pan-Hellenic Conference on Informatics*, Nov. 2018, pp.261–266. DOI: 10.1145/3291533.3291546.

[68] Bousbaci A, Kamel N. Efficient data distribution and results merging for parallel data clustering in MapReduce environment. *Applied Intelligence*, 2018, 48(8): 2408–2428. DOI: 10.1007/s10489-017-1089-7.

[69] Santhi V, Jose R. Performance analysis of parallel K-means with optimization algorithms for clustering on Spark. In *Proc. the 14th International Conference on Distributed Computing and Internet Technology*, Jan. 2018, pp.158–162. DOI: 10.1007/978-3-319-72344-0_12.

[70] Chitrakar A S, Petrović S. Efficient *k*-means using triangle inequality on spark for cyber security analytics. In *Proc. the 2019 ACM International Workshop on Security and Privacy Analytics*, Mar. 2019, pp.37–45. DOI: 10.1145/3309182.3309187.

[71] Bahmani B, Moseley B, Vattani A, Kumar R, Vassilvitskii S. Scalable k-means++. *Proceedings of the VLDB Endowment*, 2012, 5(7): 622–633. DOI: 10.14778/2180912.2180915.

[72] Shafiq M O, Torunski E. A parallel K-Medoids algorithm for clustering based on MapReduce. In *Proc. the 15th Int. Conf. Machine Learning and Applications*, Dec. 2016, pp.502–507. DOI: 10.1109/ICMLA.2016.0089.

[73] Yue X, Man W, Yue J, Liu G C. Parallel K-Medoids++ spatial clustering algorithm based on MapReduce. arXiv: 1608.06861, 2016. https://doi.org/10.48550/arXiv.1608.06861, Mar. 2024.

[74] Martino A, Rizzi A, Frattale Mascioli F M. Efficient approaches for solving the large-scale *k*-medoids problem: Towards structured data. In *Proc. the 9th International Joint Conference on Computational Intelligence*, Nov. 2017, pp.199–219. DOI: 10.1007/978-3-030-16469-0_11.

[75] Beckmann N, Kriegel H P, Schneider R, Seeger B. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. the 1990 ACM SIGMOD International Conference on Management of Data*, May 1990, pp.322–331. DOI: 10.1145/93597.98741.

[76] Goyal P, Challa J S, Kumar D, Balasubramaniam S, Goyal N. Grid-R-tree: A data structure for efficient neighborhood and nearest neighbor queries in data mining. *International Journal of Data Science and Analytics*, 2020, 10(1): 25–47. DOI: 10.1007/s41060-020-00208-2.

[77] Chen L, Gao Y J, Huang X R, Jensen C S, Zheng B L. Efficient distributed clustering algorithms on star-schema heterogeneous graphs. *IEEE Trans. Knowledge and Data Engineering*, 2022, 34(10): 4781–4796. DOI: 10.1109/TKDE.2020.3047631.

[78] Andrade G, Ramos G, Madeira D, Sachetto R, Ferreira R, Rocha L. G-DBSCAN: A GPU accelerated algorithm for density-based clustering. *Procedia Computer Science*, 2013, 18: 369–378. DOI: 10.1016/j.procs.2013.05.200.

[79] Chen C C, Chen M S. HiClus: Highly scalable density-based clustering with heterogeneous cloud. *Procedia Computer Science*, 2015, 53: 149–157. DOI: 10.1016/j.procs.2015.07.289.

[80] Hu X J, Liu L, Qiu N J, Yang D, Li M. A MapReduce-based improvement algorithm for DBSCAN. *Journal of Algorithms & Computational Technology*, 2018, 12(1): 53–61. DOI: 10.1177/1748301817735665.

[81] Gu Y H, Ye X Y, Zhang F, Du Z H, Liu R Y, Yu L F. A parallel varied density-based clustering algorithm with optimized data partition. *Journal of Spatial Science*, 2018, 63(1): 93–114. DOI: 10.1080/14498596.2017.1352542.

[82] Han D W, Agrawal A, Liao W K, Choudhary A. A novel scalable DBSCAN algorithm with Spark. In *Proc. the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops*, May 2016, pp.1393–1402. DOI: 10.1109/IPDPSW.2016.57.

[83] Huang F, Zhu Q, Zhou J, Tao J, Zhou X C, Jin D, Tan X C, Wang L Z. Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the spark platform. *Remote Sensing*, 2017, 9(12): 1301. DOI: 10.3390/rs9121301.

[84] Zhang Y F, Chen S M, Yu G. Efficient distributed density peaks for clustering large data sets in MapReduce. *IEEE Trans. Knowledge and Data Engineering*, 2016, 28(12): 3218–3230. DOI: 10.1109/TKDE.2016.2609423.

[85] Guttman A. R-trees: A dynamic index structure for spatial searching. In *Proc. the 1984 ACM SIGMOD International Conference on Management of Data*, Jun. 1984, pp.47–57. DOI: 10.1145/602259.602266.

[86] Ertöz L, Steinbach M, Kumar V. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proc. the 2003 SIAM International Conference on Data Mining*, Jan. 2003, pp.47–58. DOI: 10.1137/1.9781611972733.5.

[87] Cao Z W, Zhou Y. Parallel text clustering based on MapReduce. In *Proc. the 2nd International Conference on Cloud and Green Computing*, Nov. 2012, pp.226–229. DOI: 10.1109/CGC.2012.128.

[88] Wang S J, Eick C F. MR-SNN: Design of parallel shared nearest neighbor clustering algorithm using MapReduce. In *Proc. the 2nd International Conference on Big Data Analysis*, Mar. 2017, pp.312–315. DOI: 10.1109/ICBDA.2017.8078831.

[89] Gagolewski M, Bartoszuk M, Cena A. Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm. *Information Sciences*, 2016, 363: 8–23. DOI: 10.1016/j.ins.2016.05.003.

[90] Li X. Parallel algorithms for hierarchical clustering and cluster validity. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1990, 12(11): 1088–1092. DOI: 10.1109/34.61708.

[91] Wu C H, Horng S J, Tsai H R. Efficient parallel algorithms for hierarchical clustering on arrays with reconfigurable optical buses. *Journal of Parallel and Distributed Computing*, 2000, 60(9): 1137–1153. DOI: 10.1006/jpdc.2000.1644.

[92] Du Z, Lin F. A novel parallelization approach for hierarchical clustering. *Parallel Computing*, 2005, 31(5): 523–527. DOI: 10.1016/j.parco.2005.01.001.

[93] Johnson E, Kargupta H. Collective, hierarchical clustering from distributed, heterogeneous data. In *Proc. the 2000 Large-Scale Parallel Data Mining*, Feb. 2000, pp.221–244. DOI: 10.1007/3-540-46502-2_12.

[94] Olson C F. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 1995, 21(8): 1313–1325. DOI: 10.1016/0167-8191(95)00017-I.

[95] Dash M, Liu H, Scheuermann P, Tan K L. Fast hierarchical clustering and its validation. *Data & Knowledge Engineering*, 2003, 44(1): 109–138. DOI: 10.1016/S0169-023X(02)00138-6.

[96] Jin C, Liu R Q, Chen Z Z, Hendrix W, Agrawal A, Choudhary A. A scalable hierarchical clustering algorithm using Spark. In *Proc. the 1st Int. Conf. Big Data Computing Service and Applications*, Mar. 30–Apr. 2, 2015, pp.418–426. DOI: 10.1109/BigDataService.2015.67.

[97] Mazzeo G, Zanilo C. The parallelization of a complex hierarchical clustering algorithm: Faster unsupervised learning on larger data sets. Technical Report, University of California, Los Angeles, 2016.

[98] Wang Y, Narasayya V, He Y Y, Chaudhuri S. PACk: An efficient partition-based distributed agglomerative hierarchical clustering algorithm for deduplication. *Proceedings of the VLDB Endowment*, 2022, 15(6): 1132–1145. DOI: 10.14778/3514061.3514062.

[99] Yang J, Wang W, Wang H X, Yu P. δ-Clusters: Capturing subspace correlation in a large data set. In *Proc. the 18th Int. Conf. Data Engineering*, Feb. 26–Mar. 1, 2002, pp.517–528. DOI: 10.1109/ICDE.2002.994771.

[100] Friedman J H, Meulman J J. Clustering objects on subsets of attributes. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 2004, 66(4): 815–849. DOI: 10.1111/j.1467-9868.2004.02059.x.

[101] Domeniconi C, Papadopoulos D, Gunopulos D, Ma S. Subspace clustering of high dimensional data. In *Proc. the 2004 SIAM Int. Conf. Data Mining*, Apr. 2004, pp.517–521. DOI: 10.1137/1.9781611972740.58.

[102] Sequeira K, Zaki M. SCHISM: A new approach for inter-

esting subspace mining. In *Proc. the 4th IEEE International Conference on Data Mining*, Nov. 2004, pp.186–193. DOI: 10.1109/ICDM.2004.10099.

[103] Chang J W, Jin D S. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proc. the 2002 ACM Symposium on Applied Computing*, Mar. 2002, pp.503–507. DOI: 10.1145/508791.508886.

[104] Kailing K, Kriegel H P, Kröger P. Density-connected subspace clustering for high-dimensional data. In *Proc. the 4th SIAM International Conference on Data Mining*, Apr. 2004, pp.246–256. DOI: 10.1137/1.9781611972740.23.

[105] Kriegel H P, Kroger P, Renz M, Wurst S. A generic framework for efficient subspace clustering of high-dimensional data. In *Proc. the 5th IEEE Int. Conf. Data Mining*, Nov. 2005, pp.250–257. DOI: 10.1109/ICDM.2005.5.

[106] Assent I, Krieger R, Müller E, Seidl T. DUSC: Dimensionality unbiased subspace clustering. In *Proc. the 7th IEEE International Conference on Data Mining*, Oct. 2007, pp.409–414. DOI: 10.1109/ICDM.2007.49.

[107] Assent I, Krieger R, Müller E, Seidl T. INSCY: Indexing subspace clusters with in-process-removal of redundancy. In *Proc. the 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp.719–724. DOI: 10.1109/ICDM.2008.46.

[108] Kaur A, Datta A. A novel algorithm for fast and scalable subspace clustering of high-dimensional data. *Journal of Big Data*, 2015, 2(1): Article No. 17. DOI: 10.1186/s40537-015-0027-y.

[109] Zhu B, Mara A, Mozo A. CLUS: Parallel subspace clustering algorithm on Spark. In *Proc. the 2015 Short Papers and Workshops on New Trends in Databases and Information Systems*, Sept. 2015, pp.175–185. DOI: 10.1007/978-3-319-23201-0_20.

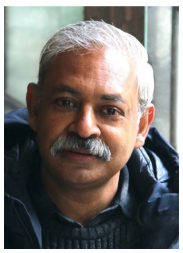
[110] Zhu B, Mozo A, Ordozgoiti B. PSCEG: An unbiased parallel subspace clustering algorithm using exact grids. In *Proc. the 24th European Symposium on Artificial Neural Networks*, Apr. 2016, pp.581–586.

[111] Gao Z P, Fan Y D, Niu K, Ying Z Y. MR-Mafia: Parallel subspace clustering algorithm based on MapReduce for large multi-dimensional datasets. In *Proc. the 2018 IEEE International Conference on Big Data and Smart Computing*, Jan. 2018, pp.257–262. DOI: 10.1109/BigComp.2018.00045.

[112] Kaul M, Yang B, Jensen C S. Building accurate 3D spatial networks to enable next generation intelligent transportation systems. In *Proc. the 14th International Conference on Mobile Data Management*, Jun. 2013, pp.137–146. DOI: 10.1109/MDM.2013.24.

[113] Springel V, White S D M, Jenkins A *et al.* Simulations of the Formation, evolution and clustering of galaxies and quasars. *Nature*, 2005, 435(1): 629–636. DOI: 10.1038/nature03597.

Jagat Sesh Challa got his M.Sc. (Tech.) degree in information systems in 2010, his M.E. degree in software system in 2012, and his Ph.D. degree in computer science in 2019, all from the Birla Institute of Technology and Science (BITS), Pilani Campus. Since Feb. 2021, he has been employed as an assistant professor at BITS Pilani. His current research interests include big data analytics, high performance computing, stream analytics, computer vision, materials informatics, human-computer interaction, and federated learning.



Navneet Goyal got his M.Sc. degree in mathematics from H.N.Bahuguna University, Srinagar (U.P) in 1988, his M.Phil. degree in mathematics in 1989 and his Ph.D. degree in applied mathematics in 1995 from the University of Roorkee (now IIT, Roorkee) Roorkee. Since 1995, he has been working as a faculty of various designations (assistant professor, associate professor, professor, senior professor) at Birla Institute of Technology and Science (BITS), Pilani Campus. Currently, he is also the head of the CSIS Department at BITS Pilani, Pilani Campus. His current research interests include big data analytics, AI/ML, earth observation, satellite image analytics, IoT data analytics, data provenance, databases and warehousing, etc.

Amogh Sharma got his B.E. degree in computer science from BITS Pilani in 2019. Since July 2019, he has been working with Uber (India and USA) as a software engineer (July 2019–Jan. 2022) and senior software engineer (since Feb 2022). His research interests include data mining and parallel algorithms.

Nikhil Sreekumar got his B.Tech. degree in CSE from TKM College of Engineering, Kollam in 2014, his M.E. degree in CS from BITS Pilani in 2016, and currently is pursuing his Ph.D. degree from the University of Minnesota. His current research interests include big data and distributed computing.

Sundar Balasubramaniam completed his B.E. degree in electronics and communication from the College of Engineering, Anna University, Chennai, in 1988, his M.Tech. degree in computer science from NIT Warangal in 1990, and his M.S. degree in computer science from Indiana University Bloomington in 1998. Since 2019, he has been working as an independent consultant and product designer for higher education in Bengaluru, India. His research interests include HPC, compilers, cloud computing, parallel algorithms, big data, etc.

Poonam Goyal is a professor in the Computer Science Department at BITS Pilani, Pilani Campus, Pilani. She completed her post-graduation in mathematics in 1989, and got her Ph.D. degree in applied mathematics in 1995, both from the University of Roorkee (now IIT, Roorkee), Roorkee, India, and her M.E. degree in software systems from the CSIS Department, BITS Pilani, Pilani Campus, in 2001. Her research has contributed to various social and scientific domains like social media analytics, multi-modal knowledge graphs, HPC for AI, etc.