Mr. Scan: A Hybrid/Hybrid Extreme Scale Density Based Clustering Algorithm

Benjamin Welton, University of Wisconsin
Barton P. Miller, University of Wisconsin

Density-based clustering algorithms are a widely-used class of data mining techniques that can find irregularly shaped clusters and cluster data without prior knowledge of the number of clusters the data contains. DBSCAN is the most well-known density-based clustering algorithm. We introduce our extension of DBSCAN, called Mr. Scan, which uses a hybrid/hybrid parallel implementation that combines the MRNet tree-based distribution network with GPU-equipped nodes. Mr. Scan avoids the problems encountered in other parallel versions of DBSCAN, such as scalability limits, reduction in output quality at large scales, and the inability to effectively process dense regions of data. Mr. Scan uses effective data partitioning and a new merging technique to allow data sets to be broken into independently processable partitions without the reduction in quality or large amount of node-to-node communication seen in other parallel versions of DBSCAN. The dense box algorithm designed as part of Mr. Scan allows for dense regions to be detected and clustered without the need to individually compare all points in these regions to one another. Mr. Scan was tested on both a geolocated Twitter dataset and image data obtained from the Sloan Digital Sky Survey. In testing Mr. Scan we performed end-to-end benchmarks measuring complete application run time from reading raw unordered input point data from the file system to writing the final clustered output to the file system. The use of end-to-end benchmarking gives a clear picture of the performance that can be expected from Mr. Scan in real world use cases. At its largest scale, Mr. Scan clustered 6.5 billion points from the Twitter dataset on 8,192 GPU nodes on Cray Titan in 7.5 minutes.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems - Distributed Applications; I.5.3 [**Pattern Recognition**]: Clustering

General Terms: Density Based Clustering, Distributed Systems, General Purpose Graphics Processors

Additional Key Words and Phrases: DBSCAN, MRNet, GPU, CUDA

ACM Reference Format:
Benjamin Welton and Barton P. Miller, 2015. Mr. Scan: A Hybrid/Hybrid Extreme Scale Density Based Clustering Algorithm. *ACM* 0, 0, Article 0 (0), 22 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Modern leadership class supercomputers tie together a diverse collection of resources, such as CPUs, GPUs, and high speed interconnects. To get maximum performance on leadership class machines, an application must efficiently use (and balance the use of) these resources. In this paper, we present Mr. Scan [Welton et al. 2013] [Welton and Miller 2014] a new extreme scale density based clustering algorithm that uses a hybrid/hybrid model to effectively utilize all the resources available on leadership class machines. The term *hybrid* computing has been used when a program incorporates multiple types of resources. A well know example of hybrid computing is the combination of distributed computing across nodes with multi-threading within a node, as exemplified by combining MPI [Lusk et al. 1996] with OpenMP [Dagum and Menon 1998] or combining MapReduce [Dean and Ghemawat 2008] with pthreads [Mueller 1993]. Another type of hybrid computing combines a general purpose CPU and one or more GPUs. We describe a computation as *hybrid/hybrid* when it combines communication, CPU, and GPU resources, with the goal to achieve efficiency at leadership-class computing scales. With the hybrid/hybrid architecture of Mr. Scan we are able to scale to multi-billion point datasets and to a large number of GPU nodes (8192 at our maximum tested scale).

Clustering is the act of classifying data points, where data points that are considered similar are contained in the same cluster and dissimilar points are in different clusters. Clustering helps researchers and data analysts gain insight into their data, e.g., identifying and tracking objects such as gamma-ray bursts in sky observation data [Davidoff and Wozniak 2003], monitoring the growth and decline of forests in the United States [Mills et al. 2011] and identifying performance bottlenecks in large-scale parallel applications [Gamblin et al. 2010]. We focus on a type of clustering algorithm called density-based clustering, which classifies points into clusters based on the density of the region surrounding the point. Density-based clustering detects the number of clusters in a dataset without prior knowledge and is able to find clusters with non-convex shapes.

Datasets such as the Sloan Digital Sky Survey [sds 2013] and geo-located tweets from Twitter [twi 2013] are useful to cluster but are too large (i.e., billions of data points) to be practically computed on a small or medium-sized parallel computer (100's to 1000's of nodes) by any non-trivial clustering algorithm. Clustering extremely large datasets requires the largest-scale parallel systems that are in use today. However, there are few distributed density-based clustering algorithms designed to run on these large-scale systems. Existing distributed density-based algorithms typically reduce the quality of the output when compared to the single-node version, or they do not scale to the sizes needed for these datasets.

Mr. Scan is our extension of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithm [Ester et al. 1996]. DBSCAN is the most widely cited density-based clustering algorithm and has been shown to be well-suited for data analysis in many fields, e.g., the analysis of laser ablated material [Sonntag et al. 2011] and tracking population movement by use of geotagged photographs [Kisilevich et al. 2010]. The benefits DBSCAN has over other clustering algorithms are that it has the ability to find irregularly shaped clusters, it distinguishes data points that are considered noise (i.e., points in low density regions) from clusters, and it is able to cluster data where the number of clusters in the dataset is not known in advance. These features come at a cost, since the computational complexity of DBSCAN is $O(n^2)$ where n is the number of points in the input dataset. This complexity is a result of the calculation of a n x n matrix containing the distances between all points. This matrix can be replaced with a spatial tree index which reduces the cost of distance calculations leading to an average case complexity of $O(n \log n)$.

Mr. Scan is the first implementation of DBSCAN that can scale efficiently to multi-billion data points and the first distributed DBSCAN algorithm that incorporates the use of GPUs. It uses a programming paradigm that organizes processes into a multi-level tree with an arbitrary topology. In this multi-level tree paradigm, DBSCAN calculations are done on the GPU leaf nodes and these results are combined on non-leaf nodes. Mr. Scan is also the first clustering algorithm to use this programming paradigm to our knowledge.

The Mr. Scan algorithm is designed to solve the end-to-end problem. The end-to-end problem is defined as the total time to go from raw unordered input on disk to final clustered output written to the file system. The usage of the end-to-end metric for measuring performance provides a complete view of performance and more closely relates to performance in real world usage of density based clustering algorithms. Using the end-to-end metric as a performance measure, we show that the multi-level tree design of Mr. Scan can cluster 6.5 billion points using 8,192 GPU nodes with a total time of 7.5 minutes.

The ability to cluster billions of data points with DBSCAN can only be realized if the key obstacles to scaling DBSCAN are overcome: load balancing, cluster merging, and distributing data advantageously. The running time of DBSCAN increases as a function of the spatial density of input data points, which causes a load imbalance when compute nodes contain regions of varying density. We modify DBSCAN to find the most dense regions and infer their membership in a cluster without evaluating the points inside these dense regions. Results from DBSCAN compute nodes must be merged accurately without requiring the entirety of each cluster. We resolve this by requiring a small, bounded number of representative points per cluster to perform a merge. Finally, data must be distributed in a manner that balances DBSCAN's clustering operation and the overhead of merging clusters. We achieve this with a heuristic that spatially decomposes the data into partitions to balance the merge overhead. Each partition contains roughly equal point counts to aid in balancing DBSCAN clustering time.

In Section 2 we describe the DBSCAN algorithm and discuss other methods that attempt to parallelize DBSCAN and other related work. Section 3 introduces the Mr. Scan algorithm and describe how it overcomes DBSCAN's scaling obstacles. Section 4 describes the experiments used to benchmark Mr. Scan using data from the microblogging service Twitter and the Sloan Digital Sky Survey. Section 5 presents and discusses the scaling results of both datasets. Finally, Section 6 presents our concluding thoughts.

2. BACKGROUND AND RELATED WORK

Due to DBSCAN's popularity among density-based clustering algorithms, optimization and parallelization of the algorithm has been widely studied [Ali et al. 2010]. We first explain the DBSCAN algorithm in detail, then present previous parallelization efforts that are most significant to the parallelization style of Mr. Scan along with the most scalable algorithms.

2.1. The DBSCAN Clustering Algorithm

DBSCAN clusters data points by density. Its notion of density comes from its two parameters known as *Eps* and *MinPts*. DBSCAN operates by finding the *Eps-neighborhood* of each point. The *Eps*-neighborhood of a point p is the set of points that are located within *Eps* distance of p. The point p is considered a core point if there are at least *MinPts* points in its *Eps*-neighborhood. All other points are classified as *non-core* points. Non-core points can have two distinctions: a *border point* or a *noise point*. A border point is a non-core point that contains at least one core point in its *Eps*-neighborhood, whereas a noise point does not.

A cluster is formed by the set of core and border points reachable from a particular core point. Once an unvisited core point is found, it is considered a new cluster along with its *Eps*-neighborhood. This cluster is expanded by finding the *Eps*-neighborhood of each point classified in the cluster until all points that are reachable from the first core point are found. For this reason, DBSCAN's clustering results can vary slightly if the order in which *Eps*-neighborhoods are discovered is changed. Figure 1 shows an example of the DBSCAN clustering process.

The performance of the DBSCAN algorithm varies greatly based on the presence (or lack thereof) of a spatial index. DBSCAN without a spatial index is $O(n^2)$ in time complexity. This is due to not limiting the amount of points compared by the distance function. Without a spatial index all points in the dataset must be compared with each other to determine which points are core. A spatial index however reduces the number of points which must be compared by limiting the search to a smaller subset of points that are in the region of the point being queried. The average case complexity improves to $O(n \log n)$ by use of a spatial index (e.g., R*-tree or KD-tree).

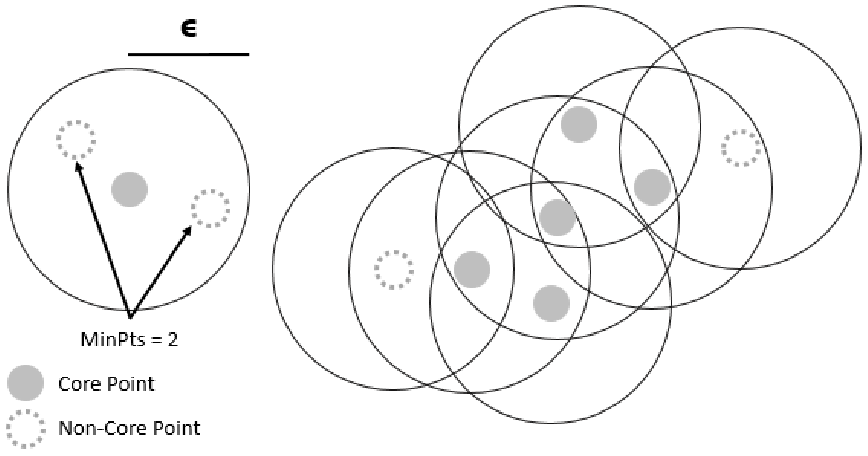


Fig. 1: DBSCAN clustering example showing classification of *core* and *non-core* points in a dataset.

2.2. Past Optimizations of DBSCAN

DBSCAN has been parallelized by multiple past projects. One of the first was PDBSCAN [Xu et al. 1999]. This algorithm used a distributed R*-tree to partition the dataset among many compute nodes. Distributed R*-trees partition data but they replicate the entire index on each node. If a neighborhood query included an area of the dataset that resides on different node, the node that started the query must send a message to obtain the data. This algorithm showed linear speedup up to 8 nodes, but the amount of messages sent grew super-linearly in most cases, which hampered its scalability. Another algorithm, DBDC [Januzaj et al. 2004], assumes that the dataset to cluster is already distributed among the compute nodes. DBDC pioneered the idea of using many slave nodes to cluster a portion of the dataset and merging the final result at a master node, and also the idea of sending a smaller number of points to represent the locally found clusters to increase scalability. This technique scaled linearly up to 30 nodes, but the manner in which representative points were picked decreased the quality of the clustering output when compared to traditional DBSCAN, and the assumption of already distributed data further degraded quality.

There have been two Map/Reduce implementations of DBSCAN, MR-DBSCAN [He et al. 2011] and DBSCAN-MR [Dai and Lin 2012]. MR-DBSCAN was able to cluster 1.9 billion points of 2D taxi-cab traces in approximately 5,800 seconds. However, the authors preprocessed the data prior to running DBSCAN to reduce the negative effects of high-density regions and did not account for this preprocessing time in their results (they did not measure end-to-end time). Also, the parameters for MR-DBSCAN's runs were chosen solely for speed and not for quality of the data analysis [He 2013]. Aside from these issues, neither of the Map/Reduce implementations showed near-linear speedup nor the ability to scale weakly and only demonstrated their algorithms on up to 12 multi-core nodes.

Recently, a distributed heuristic based approach for approximating DBSCAN has been developed called Paridcle [Patwary et al. 2014]. Paridcle uses a density based sampling approach to dramatically improve performance by limiting the number of points that need to be processed by DBSCAN. Unlike previous DBSCAN implementations that attempted a density based sampling approach, Paridcle is able to maintain a high quality for output by carefully constructing the sample for which to perform DBSCAN on. The authors show that Paridcle is capable of near linear speed up, showing a performance of 3917x while using 4096 cores. While Paridcle shows good performance, the usage of an approximate DBSCAN algorithm differs from the parallel DBSCAN approach of Mr. Scan where no quality reducing approximations are used.

Several algorithms attempted to improve the single-core performance of DBSCAN. TI-DBSCAN [Kryszkiewicz and Lasek 2010] uses the triangle inequality. The input dataset is sorted to determine a point's *Eps*-Neighborhood, which is similar to the way our GPU implementation of the algorithm uses its KD-tree. Another version of DBSCAN [Kryszkiewicz and Skonieczny 2005] attempts to remove core points early from the DBSCAN calculation. This idea is similar to Mr. Scan's dense box optimization, but their method appears that it would change the result of DBSCAN significantly, even though the authors do not comment on this effect in the paper. In comparison, Mr. Scan's dense region calculation has an extremely small impact on quality when compared to traditional DBSCAN.

2.3. The Multicast Reduction Network

The Multicast Reduction Network (MRNet) [Roth et al. 2003] is a communication framework for creating scalable applications by providing a tree based overlay network abstraction to the layout of processes in a distributed application. A tree based overlay network is ideal for scaling because it limits the amount of work any single process must perform. If the workload increases to a point where a process would be overloaded (compute or I/O bound), child nodes can be added to spread the workload among more processes (and nodes). In order for a tree based network to achieve this property the amount of data processed by each node must be identical. This means the output produced by each process in the tree should be equal to or less than the amount of data received. Achieving this goal is accomplished by having processes preform reduction operations (such as summation, discard, and other operations) on data as it passes through the node.

The MRNet framework handles the creation of the processes in the tree and the communication infrastructure between the processes. User defined reduction/aggregation and multicast filters can be supplied (in the form of shared library objects) to MRNet for use in the tree to reduce data as it moves up to the root. Users of MRNet can specify the exact topology of the tree and the placement of processes on nodes via a topology file. The MRNet framework has been used extensively to build highly scaleable tools [Schulz et al. 2005] [Arnold et al. 2007] [Ahn et al. 2009] and is used by Mr. Scan to achieve good scalability on leadership class machines.

3. THE MR. SCAN ALGORITHM

Mr. Scan is a hybrid/hybrid implementation of the DBSCAN algorithm with four phases: partition, cluster, merge, and sweep. Mr. Scan starts with a single input file on a parallel file system and writes as output a file of the points included in a cluster and their cluster IDs as output. The input points are contained in a single binary or text file. Each input point has a unique ID number, coordinates, and an optional weight that can be used for analysis of the clustered output. Figure 2 gives an overview of the Mr. Scan algorithm. All four phases of the Mr. Scan algorithm take place using the same tree layout of processes and the layout of the tree (number of leaf nodes and fanout) is user definable.

In the partition phase, the input file is read by the leaf nodes of the partitioner. The partitioner is responsible for creating one partition per clustering process (one partition per leaf node) from a given input file. The input file can contain billions of points

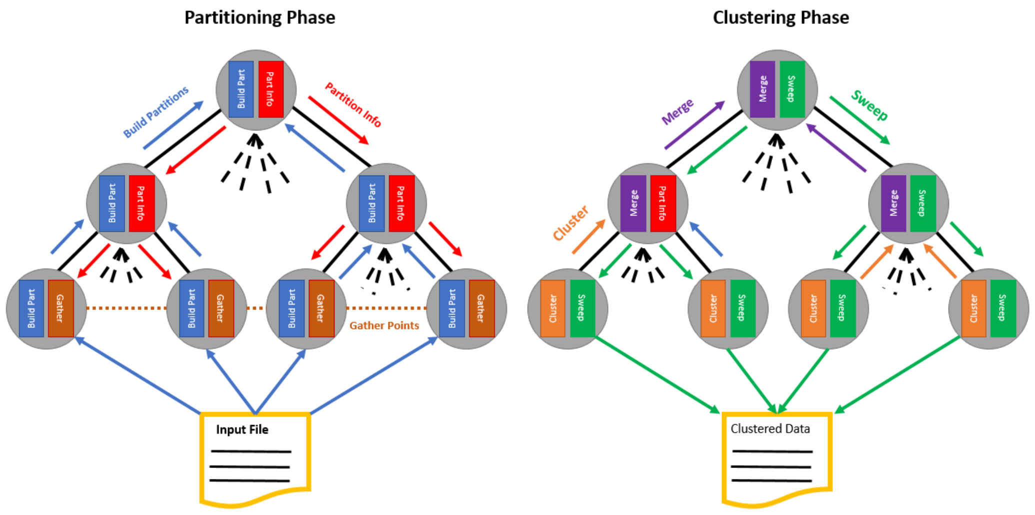


Fig. 2: Overview of the Mr. Scan algorithm

and reach sizes up to 300 GB, so the partitioner is distributed using MRNet [Roth et al. 2003] to parallelize this step. Each worker process of the partitioner moves the completed partitions (via message passing) to the node responsible for processing that partition. The cluster phase begins after all leaf nodes have received the completed partitions they are responsible for processing. Each Mr. Scan leaf process clusters its assigned partition using our GPU version of DBSCAN and picks a small, constant set of points to represent each cluster. The representative points are sent to the intermediate processes to start the merge phase where the clusters are progressively merged by each level of intermediate processes until they reach the root. The root performs the final merge and assigns a global ID to each cluster. Mr. Scan then starts the sweep phase, and sends the global cluster IDs down the tree, where each point is identified with its correct global cluster ID and written to the output file in parallel by the leaf processes.

In this section, we describe the design of each of Mr. Scan's phases, and how they solve the three challenges in scaling DBSCAN: load imbalance, distributed merge, and data distribution.

3.1. Partitioner

In addition to the basic goal of dividing an input dataset into n partitions given n leaf processes, we have three main goals for the design of Mr. Scan's partitioner. First and most important, the partitioner must produce partitions capable of yielding a correct DBSCAN result when clustered and merged. Second, the output partitions must have roughly equal computational costs when being clustered. The partitioner does not need to produce perfectly balanced partitions, since the dense box optimization described in Section 3.2.3 plays a large role in controlling load balance. However, the partitioner does hold some responsibility for controlling the load balance during the cluster phase. Third, the partitioner must perform well enough to avoid becoming a significant portion of Mr. Scan's overall time, especially as the size of the input dataset grows. This means as few operations as possible should be I/O bound, and leads to the design decision of distributing the partitioner among many nodes. We will discuss how we meet these three goals below.

3.1.1. Correctness. We define a correct partitioning as a set of partitions that merge to form a global clustering that is equivalent to executing non-parallel DBSCAN on the entire input dataset. It is impossible for this definition to be satisfied when the partitions each contain a disjoint subset of the input dataset, because any point whose *Eps*-neighborhood includes points in a different partition than its own would return an incomplete *Eps*-neighborhood [Xu et al. 1999] [He et al. 2011] [Dai and Lin 2012]. To address this, we add a shadow region to each partition. The *shadow region* is the set of points not already included in the partition that lie *Eps* distance from the partition's boundary. A *shadow point* is a point that lies in a shadow region with respect to a partition, and a *partition point* is a point already included in the partition. When the shadow region is added to a partition, each partition point's *Eps*-neighborhood contains only partition points or shadow points, and thus is complete within the partition.

3.1.2. Partitioning Algorithm. The second goal of the partitioner is to control load balance in the cluster phase by creating computationally equivalent partitions. Therefore, we must have a way to estimate a partition's computational cost to DBSCAN. Mr. Scan uses the partition's point count for this estimation. We have established in Section 1 that DBSCAN's performance is largely dependent on the spatial density of points and not pure point count, so point count is not an ideal measure for an unmodified DBSCAN implementation. We use point count instead of density because our modified DBSCAN's performance is positively impacted by density, so point count is a more accurate measure in this case.

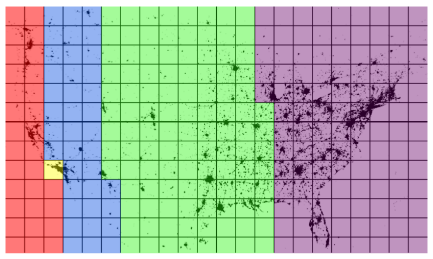
A DBSCAN partitioning algorithm must output DBSCAN partitions that are not only correct, but profitable. We denote a partition as profitable if it meets two constraints. The first is that the longest distance across the partition must be greater than *Eps*. If the distance is less than *Eps*, the *Eps*-neighborhood of each point is guaranteed to include each point in the partition, and there is no need to invoke DBSCAN. The second constraint is that each partition must contain at least *MinPts* points. Otherwise, we would already know that every point is a noise point, and DBSCAN is not needed.

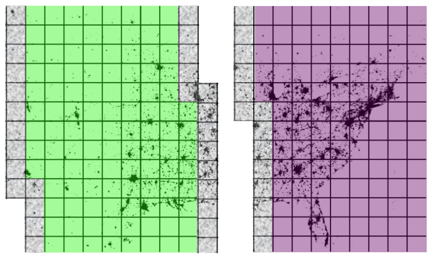
We fulfill the first constraint in our algorithm by constructing the input dataset as a grid where each cell of the grid is the same size. The minimum size of each grid cell must be *Eps* x *Eps* to ensure that each partition's longest distance (i.e. the diagonal) across is greater than *Eps*. With this restriction, the shadow region for each partition becomes the set of grid neighbors not already in the partition. The optimal selection of grid cell size is *Eps* x *Eps* for load balancing purposes. However extremely sparse datasets containing a large number of grid cells can use a larger grid cell size to reduce partition computation and memory requirements without reducing quality.

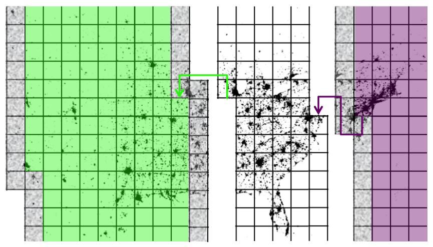
The partitioning algorithm starts by setting the *target size* of the partitions, which is an equal share of the input points. Since our partitioning algorithm forms partitions from regular grid cells that contain varying amounts of points, it is generally not possible to form partitions that are even roughly similar in their point counts when partitioning non-uniform data. Large grid cells do not pose a problem for load balancing in Mr. Scan because of our dense box optimization described in Section 3.2.3.

For simplicity, we describe the algorithm for forming partitions assuming that the input dataset's grid is 2D, however it can be extended to an arbitrary dimension. We iterate over all cells in the input grid first along the y axis, and then along the x axis. Partitions are formed sequentially through this iteration. Grid cells are added to a partition until the addition of the cell would cause the partition to exceed the initial target size. The only time that a grid cell will be added to a partition to make it exceed the target size is if the partition is the final partition formed or the partition does not yet contain any grid cells. Given the existence of grid cells that are larger than the target size, we must ensure that DBSCAN's second partitioning constraint is met: that every partition is greater than *MinPts* points. To meet this constraint, we keep track of the running difference of each partition's size from the target size. If this difference is positive, we form partitions proportionately smaller until the difference is neutral or negative again, keeping the minimum partition size set to *MinPts*. Once we finish partitioning grid cells, we add the correct shadow region to the partition, as shown in Figure 3b.

It is common in practice for the last partition to be much larger than most of the other partitions, because as the original partitions are formed, they are kept below the target size. The collective point difference of all partitions from the target size is then left for the final partition, resulting in the need for a rebalancing phase of the partitioning algorithm. Figure 3a demonstrates this, as the populous Eastern United States is included entirely in the last partition formed. Furthermore, the addition of the shadow regions increases the total number of points in the partitioned dataset, and also is likely to negatively affect whatever equality was established by the first iteration through the grid cells. Because of the increase in total points, we update the target size to the *final target size*, which is the mean of the point counts of all the partitions including shadow regions. Then, starting at the last partition formed we remove a grid cell, update the shadow region, and repeat until a specified threshold size is reached. The threshold is set to $1.075 \times finaltargetsize$ because it worked well in practice on our datasets. The removed grid cells are then added to the second-last partition formed, as in Figure 3c. This process is repeated for each partition, working sequentially backward through the partitions until we reach the first.


(a) Partition boundaries before rebalancing.


(b) Last two partitions after adding shadow regions


(c) The first step of rebalancing. We remove grid cells from the last partition formed until a threshold is reached and update its shadow region. The removed cells are added to the second-last partition, which gets rebalanced the same way.

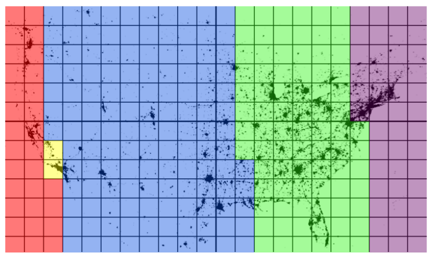

(d) Partitions after rebalancing finishes (shadow regions not shown).

Fig. 3: Mr. Scan's partition algorithm

3.1.3. Distributed Partitioner. The distributed partitioner is implemented using MRNet and uses the same MRNet tree as all other phases. The partitioner is the first phase that is run after the MRNet tree is formed. Completed paritions formed by the paritioner are directly passed to the leaf nodes responsible for performing DBSCAN on those paritions. In the original implementation of Mr. Scan the distributed parititoner was a separate MRNet process which wrote completed partitions back to the file system to be read by a separate clustering process. The usage of a separate process for partitioning and clustering was originally done for simplicity of implementation but with enormous performance costs.

The strategy for distributing the partitioner is based on the fact that the algorithm for forming partitions does not use information about each individual point. The only information needed is a grid of *Eps* x *Eps* cells and the point count for each cell. Therefore, the partitioner is able to distribute the entire input dataset across the memory of the leaf processes and only send a point count of each non-empty *Eps* x *Eps* cell to the root. The root then serially executes the algorithm described in Section 3.1.2 to determine the boundaries of each partition and broadcasts the boundaries to the leaves. The leaves then pass the completed partition via message passing to the leaf node that will be responsible for clustering that partition. The leaf responsible for that partition is determined by matching the MRNet rank (a unique identifier) assigned to each leaf node to the parititon number given to each partition by the root process.

3.2. Clustering Phase

The clustering phase runs in parallel on each leaf node, executing a highly multithreaded implementation of DBSCAN that executes on a GPU. The GPU algorithm developed for Mr. Scan is an extension of the CUDA-DClust algorithm [Böhm et al. 2009], adding two key modifications to increase scalability both at the clustering and merge steps. The main contribution of these extensions is a reduction of run-time variability caused by differing point density. We start with an overview of the CUDA-DClust algorithm in Section 3.2.1. Our two extensions to CUDA-DClust, improving the host-GPU interaction and Dense Box point elimination, are described in Sections 3.2.2 and 3.2.3.

3.2.1. The CUDA-DClust algorithm. The design of the CUDA-DClust algorithm is conceptually similar to the DBSCAN implementation described in Section 2.1. Clustering in CUDA-DClust differs from DBSCAN in that multiple DBSCAN operations to take place on the dataset simultaneously. The number of DBSCAN operations running concurrently is determined by the number of GPU blocks. A GPU block is the CUDA term for the logical grouping of threads running on a multiprocessor. Figure 4 shows CUDA-DClust at the GPU block-level.

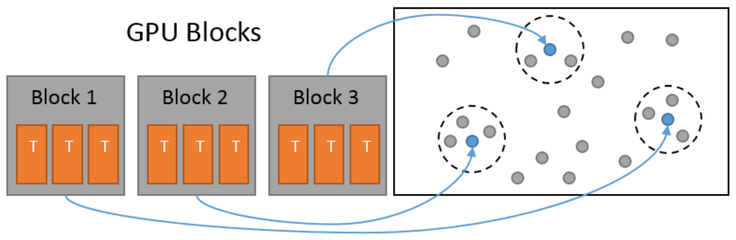


Fig. 4: CUDA-DClust GPU blocks expanding seed points.

Each GPU block is assigned a single seed point, which is a point that has not yet been expanded. The DBSCAN algorithm is then run on this point to expand the cluster and find neighboring points. CUDA-DClust uses a modified KD-tree to help DBSCAN determine possible neighbor points. The difference between a standard KD-tree and the CUDA-DClust modified KD-tree is that a leaf represents a region of points instead of a single point. The use of a KD-tree reduces the cost of point expansion by limiting the number of neighbors that need to be checked to the points in the same region of the point being expanded. Other indexing structures, such as the R*-tree typically used in a CPU implementation of DBSCAN, cannot be used on the GPU due to the overhead of traversing a tree of arbitrary depth.

After expansion, if the point is determined to be a core point, it is marked as being a member of a cluster and all of its neighbors are added to the block's queue for expansion on subsequent DBSCAN iterations. Otherwise the point is marked as noise.

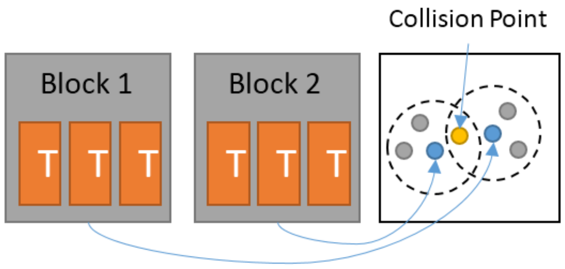


Fig. 5: Collision between two concurrently running blocks.

After all blocks have completed the expansion of their respective seed points, control is transferred back to the CPU. The CPU then copies the current state for each block from the GPU, checks for collisions between blocks, and re-seeds the blocks that have an empty point queue with a new unprocessed point. A collision occurs between two blocks when one block attempts to expand a point that has already been expanded by another block or if the point is already in the queue to be expanded by another block. Figure 5 shows an example of this collision where two blocks share a point after an expansion. These collisions indicate that the clusters being expanded in two different GPU blocks are actually the same cluster. Collisions between blocks must be tracked and corrected by merging the clusters expanded by all blocks that collide.

3.2.2. Limiting Host to GPU interaction. Synchronous memory transfers between CPU and GPU are costly operations which should be kept to a minimum. CUDA-DClust has the negative property of performing at least two memory operations between the host and GPU after every DBSCAN iteration. This results in a total of $2 \times (Points/BlockCount)$ copy operations between the host and GPU. Since leaf nodes in Mr. Scan may have widely varying point counts due to differing shadow area densities, nodes that have high point counts can take much longer to cluster. We modified CUDA-DClust so that there is only a single round trip memory operation regardless of point and block count (copying raw input to the GPU and retrieving the clustered result from the GPU).

In our new clustering algorithm, there are now two passes over the point data. Pass one classifies all the core points in the dataset and uses a method similar to CUDA-DClust to expand points. One difference is that points are not placed into a block's queue if the number of neighbor points is greater than *MinPts*, and expansion during this phase stops as soon as *MinPts* is reached. Instead of requiring a synchronous memory copy after every input seed point is expanded, the next input seed point for DBSCAN is determined by the parameters of the CUDA kernel call. This allows for all kernel invocations needed to cluster the dataset to be issued in bulk without any intervening memory copies.

The second pass expands the core points found in the first pass to generate the clusters. The clusters are found by running the same operation as above on only the core points in the dataset. When a point is expanded, all of that points neighbors are marked as being members of the cluster. When a collision occurs between two expanded clusters, the collision between these clusters is marked and rectified on the CPU after all points in the dataset has been classified. When all points have been classified, the CPU merges clusters that have collided and the final clusters are revealed.

3.2.3. Dense Box Algorithm. The dense box algorithm allows for points in dense data regions to be marked as members of a cluster without incurring the cost of expanding each point individually. Dense regions of data are detected by using the sub-divided point space generated by the modified KD-tree described in Section 3.2.1. All points in a sub-division with *dimension size* less than or equal to $\frac{Eps}{2\sqrt{2}}$ *by* $\frac{Eps}{2\sqrt{2}}$ and $pointcount \geq MinPts$ will be marked as members of a cluster. The points that are marked as being members of a cluster are not expanded when they are encountered by DBSCAN.

Since we are using an existing sub-division of the point space, there is little added complexity for detecting and eliminating dense boxes. The worst case complexity of this algorithm is $O(l)$ where l is the number of sub-divisions. In return we see a reduction in the complexity of DBSCAN. Worst case DBSCAN complexity drops from $O(n^2)$ to $O((n-p)^2)$ where *p* is the number of points eliminated by dense box. Average case run time for DBSCAN drops from $O(nlog(n))$ to $O((n-p)log(n))$. We see that as the complexity of DBSCAN rises, the number of points *p* removed by dense box increases.

3.3. Merge Algorithm

Merging clusters generated by multiple nodes is needed to generate the final output clustering. The merging process is not trivial because a single cluster may span multiple nodes and these clusters only merge if they have a core point in common. Mr. Scan's merge algorithm detects and merges clusters with overlapping core points quickly without requiring the presence of the entire clustered output. Using the entire clustered output would exhaust computational and memory limits as the output grows in size so we select a fixed number of points per grid cell (eight points) to represent the cluster's core points. The points selected to represent the core points are called representative points and are described in Section 3.3.1. The representative points and the set of non-core points of the cluster are used for merging in a method described in Section 3.3.2.

3.3.1. Selection of Representative Points. The set of representative points is the minimum set of core points from a single cluster that can correctly detect a merge inside a single grid cell. Clusters that have overlapping core points need to have at least one core point of overlap within the collective *Eps*-neighborhood that is formed by the set of representative points of the grid cell. We have determined that eight points can represent the core points of a grid cell of arbitrary density. The eight selected representative points are the points closest to the center of the sides of the grid cell and the corners of the grid cell. Figure 6 shows that when two clusters have an overlapping core point in a grid cell that at least one will be within the *Eps*-neighborhood of a representative point.

3.3.2. Merging. The merge algorithm is run on all clusters with overlapping grid cells. At this point in the algorithm, all clusters are composed of grid cells with each grid cell containing a set of representative points and the set of non-core points. The merge

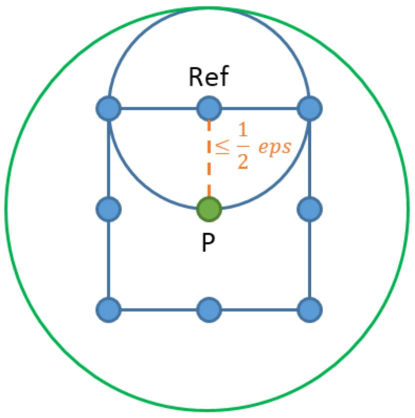


Fig. 6: Any overlapping core point P must be within $\frac{1}{2} \times Eps$ of at least one corner or side of the grid cell. We label this point *Ref*. This means that the representative point for *Ref* must fall within a $\frac{1}{2} \times Eps$-neighborhood of *Ref*. Since this entire region (shown as the blue circle) is contained in the *Eps*-neighborhood of P this means that P is always within *Eps* of a representative point.

operation is done on every pair of overlapping grid cells between two clusters. There are three types of grid cell overlaps that the merge operation must be able to handle.

The first type, a core point overlap, is when both clusters marked the same point as a core point (shown in Figure 7). Whenever a core point overlap occurs we know the two clusters merge. This case is detected if any representative point from one cluster falls within *Eps* of a representative point from the other cluster.

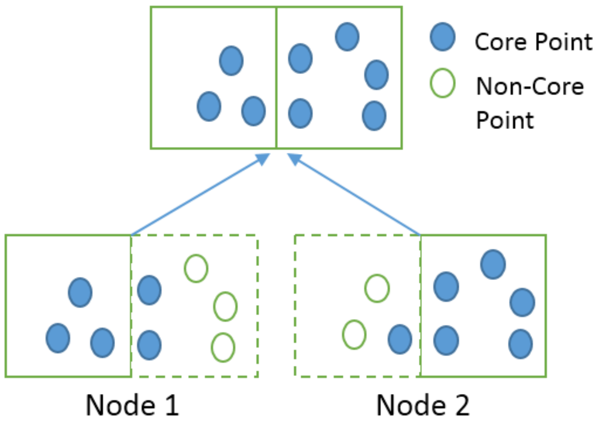


Fig. 7: Example of a core point overlap between clusters detected on different nodes.

The second type, a non-core/core overlap, is when one cluster detects a point as being core and another cluster in the grid cell detects the same point as being non-core. This case (shown in Figure 8) arises in Mr. Scan because shadow cells by definition do not have a complete set of neighbor cells on a node. We have shown in Section 3.1.1 the partitioner guarantees that all points within *Eps* of any point in the non-shadow region will be included in the partition, however this is not true of shadow regions. Points in shadow regions could have neighbors that are not visible to the node causing a misclassification of the point. We take advantage of the fact that a non-shadow region will always have the correct classification for points in its cell to detect and merge clusters that have a non-core/core overlap. If we obtain the difference between the set of non-core points found by the shadow region and the set of non-core points found by the non-shadow region we get a set of points that is unique to the shadow region. If any point in the resulting set is within *Eps* of any representative point in the non-shadow region, the clusters merge.

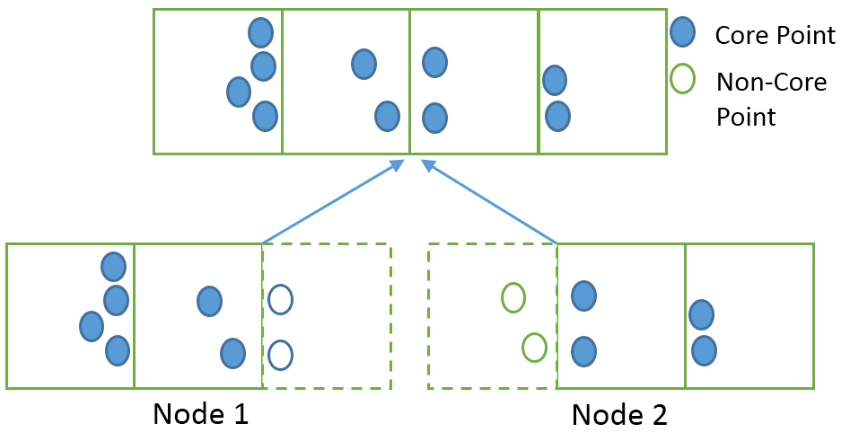


Fig. 8: Example of the second type of merge where a core point detected by one node is being detected as a non-core point in a neighboring node. This case is caused by a shadow region not having all neighbor points available (furthest left and furthest right grid cells shown).

The third type, a non-core/non-core overlap, is when two clusters do not merge but have overlapping non-core points. We detect this case so that we can remove duplicate non-core points from the output data. Since the non-core points are members of both clusters we can remove the duplicate non-core points from either of the clusters. We resolve this case by removing all duplicate non-core points from the shadow region.

When a cluster has been completed (all overlapping shadow regions have been merged) the cluster and its representative points are left on the node that performed the last merge. A summary of the cluster (point count and a cluster identifier) is used in place of the representative points. The usage of a summary for completed clusters reduces the amount of data moved to merging processes closer to the root of the tree.

3.4. Sweep Step

The purpose of the sweep step is to write the finalized clusters out to the file system. The sweep step starts with the results of the completed merge operation. It first calculates file offsets to be used by the leaf nodes to write out the points for each cluster. Next a globally unique identifier is assigned to each cluster. This information is then sent down the tree with each level of the tree reversing the merge operation. When a leaf node receives the unique identifier labeling information, it writes all points present in those clusters out to the file system.

4. EXPERIMENT SETUP

We have two goals in evaluating Mr. Scan. The first goal is to test our ability to run DBSCAN on datasets that are several billion points in size in a reasonable amount of time. These datasets must represent real-world problems, and our experiments must use DBSCAN parameters that are useful to the problem. Datasets of this size have not been successfully clustered with any density-based clustering algorithm. The second goal is to evaluate whether Mr. Scan exhibits good scaling properties. This is a difficult proposition because memory limits make comparison to a single node implementation impossible. We first evaluate weak scaling, where each leaf process is responsible for roughly 800,000 points. We then evaluate strong scaling, comparing performance at scale to the smallest Mr. Scan instantiation that memory limits allowed on our largest dataset.

We tested Mr. Scan with a synthetically generated dataset derived from geo-located tweets from Twitter and an image dataset obtained from the Sloan Digital Sky Survey. All experiments were run on the Titan supercomputer located at Oak Ridge National Laboratory. Titan is a Cray XK7 system with 18,688 compute nodes. Each compute node has sixteen 2.2GHz AMD Opteron processors with 32 GB of memory, and an NVIDIA Tesla K20 accelerator with 6 GB of memory. Titan is connected to a large Lustre file system. When we ran our experiments, only 8,972 compute nodes were available.

4.1. Twitter Experiment

Research that analyzes user activity on social media sites like Twitter and Facebook is rapidly increasing in popularity. Twitter has been used to detect and predict flu outbreaks [Lampos and Cristianini 2012], alcohol consumption [Culotta 2013], rainfall [Lampos and Cristianini 2012], overall mood of a nation [Lansdall-Welfare et al. 2012], political biases [Lampos 2012], and popular topics [Karandikar 2010]. The importance of this research is evidenced by the United States Library of Congress archiving all Twitter tweets from 2006 to the present [Telegraph 2013]. Facebook is also used for social science research [Wilson et al. 2012], and Flickr has been used to analyze popular places in a city using photo location data [Kisilevich et al. 2010]. Many of these research efforts lacked the ability to add location-based information to large scale analyses, so Mr. Scan should make large scale analysis using location information from social media more feasible.

To test Mr. Scan's ability to cluster these datasets, we collected a set of 8,519,781 geo-located tweets from Twitter's public API between August 11-21, 2012. We then used the distribution of these tweets to generate random datasets of arbitrary size for ease of experimentation. In our experiments, we used latitude and longitude as 2D Cartesian Coordinates and fixed our *Eps* value at 0.1 degree to represent a fine-grained analysis. We tested four different *MinPts* values: 4, 40, 400, and 4000 to represent a wide range of output densities.

4.2. Sloan Digital Sky Survey Experiment

The Sloan Digital Sky Survey (Sky Survey) is an imaging survey that is used to map and catalog astronomical objects using images obtained from terrestrial based telescopes [sds 2013]. The images generated by the telescopes in the survey may contain hundreds of new objects which need to be classified and cataloged. Since there are tens of thousands of such images, an automated process is needed to detect these objects. The DBSCAN algorithm has been used to automate the process of detecting, tracking, and classifying objects obtained from terrestrial based telescopes [Davidoff and Wozniak 2003] [Patwary et al. 2012]. As data sizes in grow, automated cataloging (and re-cataloging) of these datasets will be needed. Testing was done on the Baryon Oscillation Spectroscopic Survey γ frame photo object data released by SDSS in Data Release 9 [bos 2013].

5. EVALUATION

We present results from the evaluation of Mr. Scan for both the Twitter and Sky Survey datasets. In Section 5.1 both the weak and strong scaling results are presented for the Twitter dataset. A breakdown of the running time for each portion of the Mr. Scan algorithm is also provided and discussed. Section 5.2 contains the weak scaling results for the sky survey dataset.

5.1. Twitter Experiment

We evaluated both the weak and strong scaling properties of Mr. Scan using our Twitter datasets. We tested weak scaling by clustering a fixed number of points per leaf node and increasing the data size proportionally to the number of leaf nodes. In evaluating Mr. Scan, we performed testing with both a stand alone file system based partitioner [Welton et al. 2013] and a direct message passing partitioner [Welton and Miller 2014]. The difference between these two partitioning methods is the stand alone partitioner uses the file system to store the partitioned datasets before DBSCAN execution. The Mr. Scan DBSCAN process would then read the partitioned dataset and perform the clustering operation. The message passing partitioner cuts out the intermediate file system storage opting to directly pass the completed partitions to the DBSCAN clustering node. Table I describes the configurations of nodes and data sizes tested for weak scaling. We tested strong scaling by clustering our largest dataset of 6.5 billion points, starting at the number of leaf nodes that had sufficient memory to support their partition size. We then increase the node count to the highest amount the machine allowed. Finally, we evaluated the quality of Mr. Scan's output compared to a single CPU implementation of DBSCAN.

# of points	# of MRNet internal processes	# of leaves	# of partition nodes (stand alone)
102,400,000	4	128	16
204,800,000	8	256	32
409,600,000	16	512	32
1,638,400,000	32	2048	64
3,276,800,000	64	4096	96
6,553,600,000	128	8192	128

Table I: Configurations used in weak scaling experiment with partitioning node count for experiments with a stand alone partitioner

For all experiments, we use a fixed *Eps* value of 0.1 degree. MRNet uses trees with one compute node per process because there is one GPU per compute node on Titan. We use tree topologies that aim to decrease the amount of non-leaf processes in the allocation. Each topology has at most three levels, and each intermediate process has at most a 256-way fanout of child processes. In experiments where the stand alone partitioner was used, number of nodes used for the partitioner for each run was determined by selecting the best performing configuration from a prior experiment. In experiments with a message passing partitioner, all DBSCAN nodes performed partitioning.

5.1.1. Mr. Scan with Message Passing. We first present the end-to-end total time of all Mr. Scan phases when using a message passing partitioner in Figure 9. The end-to-end total time includes all portions of the algorithm from reading the raw unordered data on the file system to writing the final clustered output. A breakdown the of the performance of the partitioning phase, GPU DBSCAN operation, and the combined runtime of the DBSCAN cluster/merge/sweep phases can be seen in Figures 10, 11, and 12 respectively. The weak scaling benchmarks of Mr. Scan used four different *MinPts* values. The *MinPts* values used in the benchmark are chosen to show the impact on runtime the dense box algorithm described in Section 3.2.3. In addition, we also present strong scaling numbers for Mr. Scan shown in Figure 13.

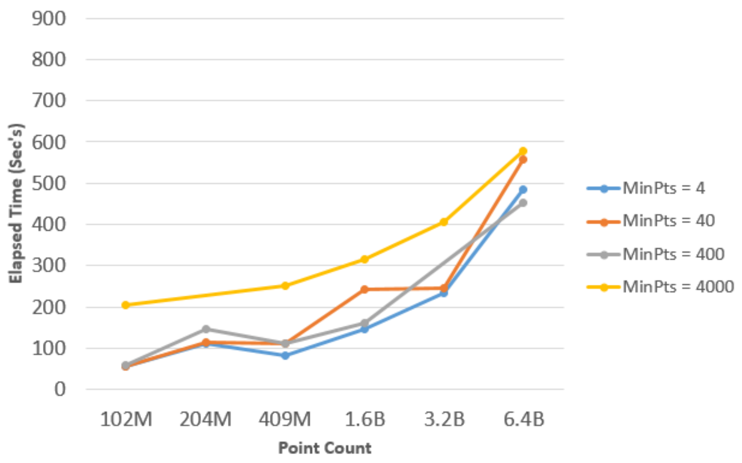


Fig. 9: Elapsed time of Mr. Scan with a message passing partitioner for the configurations listed in Table I. *Eps*=0.1 and *MinPts* varies as indicated.

Mr. Scan is able to successfully cluster 6.5 billion points with 8192 GPU equipped nodes. Figure 9 shows that this can accomplished between 453 and 576 seconds depending on the *MinPts* parameter used. We see that Mr. Scan weak scales linearly with a relatively gentle slope: as the data increases by 64x (from 102 million to 6.5 billion points) we see a total elapsed time growth of 9.8x. While we see reasonable scaling properties from Mr. Scan, the growth in elapsed time in the weak scaling experiment is less than the ideal of a flat scaling curve. There are three linear components of Mr. Scan that contribute to the reduced performance in the weak scaling experiment. Partitioning, writing the cluster output, and a slight increase in MRNet startup costs at larger node counts account for a vast majority of the increasing runtimes between node counts.

Partitioning is a small contributor to the in linearly increasing runtimes in the weak scaling experiment at our largest scale. Figure 10 shows the partitioning time for the weak scaling experiments. The majority of the impact of larger data sizes on partitioning can be attributed to reading the file containing the points to be clustered from the Lustre file system. At our largest scale of 8192 nodes, 28% of Mr. Scan's total run time was spent in the partitioning phase. 85% of the partitioning run time is reading the dataset from Lustre. In cases where Mr. Scan is not run on live data (data already on node), the cost of reading large files from the file system is unavoidable. While there are methods, such as I/O forwarders [Ali et al. 2009], that can help reduce the slope of

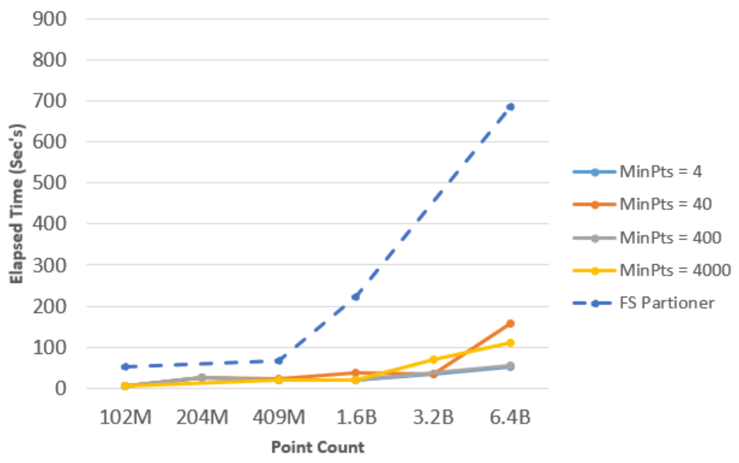


Fig. 10: Partitioning time for Mr. Scan with the message passing partitioner at different *MinPts* parameters with a comparison to a older file system based approach to partitioning

the linear curve associated with file reading there is no current method to eliminate this linear growth at large scales.

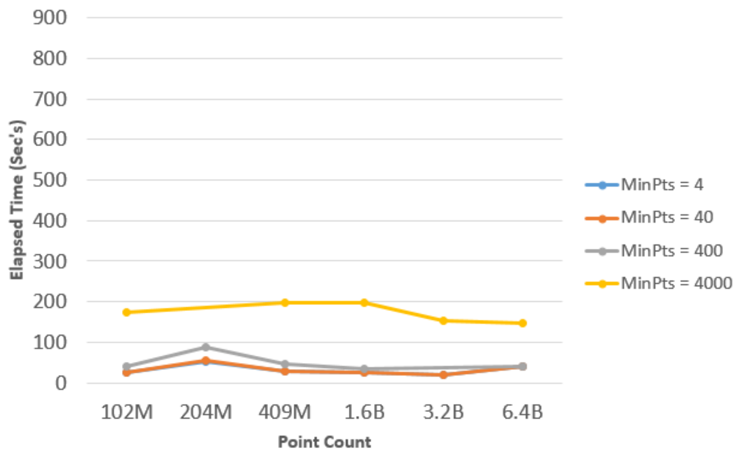


Fig. 11: GPU DBSCAN time for Mr. Scan with the message based partitioner

The partitioning phase, while having some weaknesses, is much improved over our original file system based version. The original file system partitioner used an intermediate file to move data to DBSCAN compute nodes for processing. This intermediate non-sequential write substantially increased run time for Mr. Scan and the partitioning phase in particular. The partitioning phase in the old approach comprised 68% of total run time nearly doubling the run time of Mr. Scan. As a comparison to the message passing partitioner, a representative result from the file system partitoner of Mr. Scan is included in Figure 10 (the other file system partitoner results are similar to the representative).

The writing of completed clusters to the Lustre file system is the major cause of Mr. Scan's linear growth with the message passing partitioner. Cluster output is a component of the cluster/merger/sweep phase shown in Figure 12. The time to write cluster output quickly grows from 10% of the total run time of the cluster/merge/sweep step at scales below 1.6 billion points to over 70% at scales above 1.6 billion points. The reason for the rise in run time at scale is due to the larger number of points each cluster contains with the larger datasets and the increasing number of nodes that write results to the Lustre file system. This cost can be mitigated in cases where Mr. Scan is passing the clustered output to another application that will run subsequently on the same node.

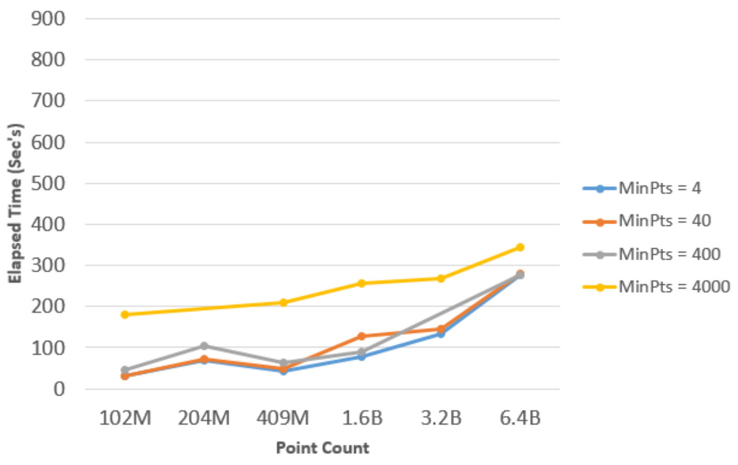


Fig. 12: Runtime of the combined Cluster, Merge, and Sweep step phases of Mr. Scan with the message based partitioner

The final linear property of Mr. Scan is the startup costs associated with increasing node counts. The cost of application startup and initialization was one of the one of the more surprising elements uncovered in our testing of Mr. Scan. Application start-up consumed approximately 10% of the total run time of Mr. Scan at the largest scale tested. The time consumed by start-up comes from two sources: a file system storm resulting from all nodes attempting to simultaneously load Mr. Scans dynamically-linked shared objects from the file system, and zeroing the GPU memory before execution of a user binary. While static linking could solve the first problem, Cray machines (such as Cray Titan where Mr. Scan was benchmarked) do not allow use of static linking for some critical libraries such as MPI and CUDA. Later versions of the Cray operating system than we had available to us (version 5.1) have methods that may reduce the penalty of loading shared libraries. The initialization problem comes from the system zeroing the GPU memory on the nodes in a user's allocation when the application is first launched. This security features is set by system administrators as a system-wide policy, and is not modifiable by the users.

The strong scaling experiment consisted of running the 6.5 billion point dataset at scales between 512 to 8192 nodes. Figure 13 shows the strong scaling results using the *MinPts* parameter of 400. The performance of Mr. Scan improves at first but at node counts greater than 2048 there is a reduction in performance. At 2048 nodes we see an improvement of 57% when compared to the total run time at 512 nodes while at 8192 nodes we see an improvement of 54%. The reason for this drop off in performance

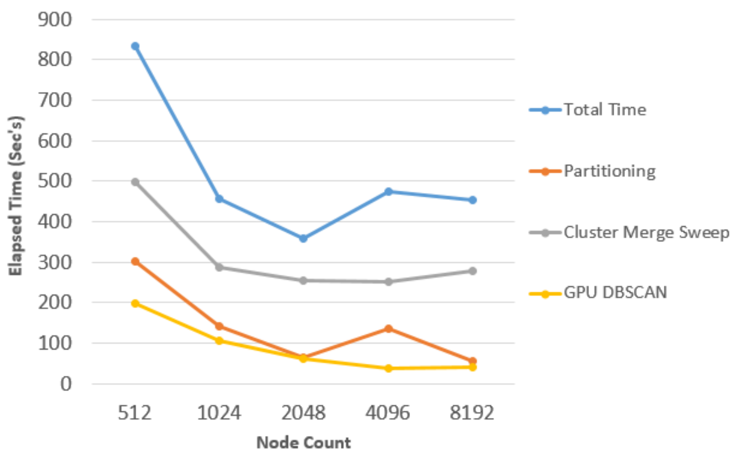


Fig. 13: Strong scaling of 6.5 billion points using the message based partitioner

is similar to the weak scaling experiments above and can be mostly attributed to file system I/O time and start up costs.

5.1.2. Quality. We evaluated the quality of Mr. Scan's output in comparison to DBSCAN on a single CPU. Quality is measured with a metric defined by the authors of DBDC [Januzaj et al. 2004]. The metric assigns a quality score between 0 and 1 to each point as $\frac{|A \cap B|}{|A \cup B|}$, where A is the cluster the point belongs to in DBSCAN's output, and B is the equivalent cluster from Mr. Scan's output. If a point is misidentified as a noise or non-noise point, it gets a quality score of 0. The final quality score is an average of the points' quality scores. Therefore, this metric is maximized when all clusters found contain the exact same points in the output, and when all noise points are identical as well. We were limited to 12.8 million points for this experiment by the memory of a single compute node. We used ELKI 0.4.1 [Achtert et al. 2011] as our reference DBSCAN implementation, which took over 35 hours to cluster the 12.8 million points. Figure 14 shows near-perfect quality at the scales we were able to test, as Mr. Scan did not get lower than a .995 quality score.

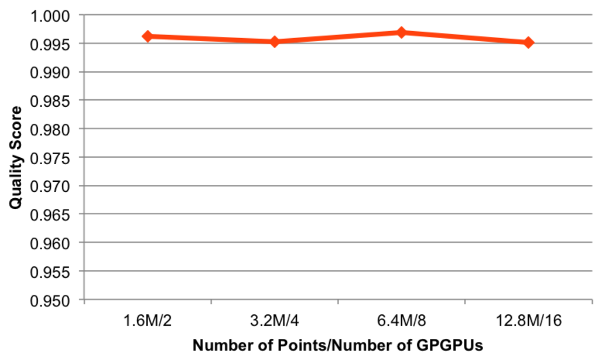


Fig. 14: Quality measurement of Mr. Scan on the Twitter dataset

5.2. Sky Survey Experiment

The sky survey experiment consisted of a weak scaling experiment with a maximum point count of 1.6 billion points processed on 2048 nodes. This experiment was run with a fixed *Eps* value of 0.00015 and a fixed *MinPts* of 5. Figure 15 shows the weak scaling results for the sky survey experiment with the message passing and file system based partitioners.

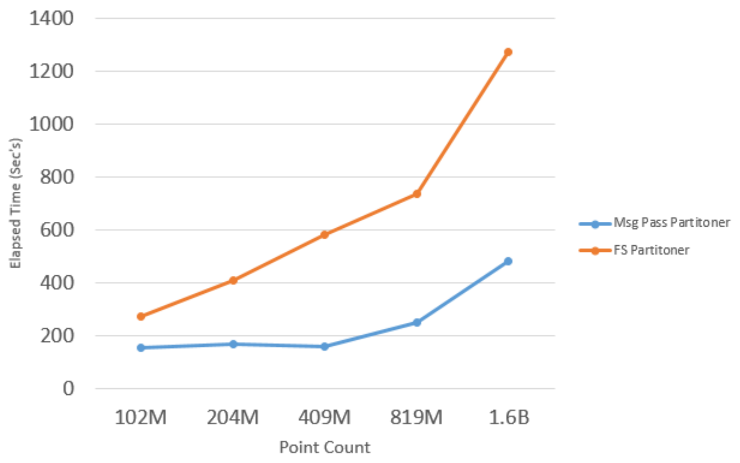


Fig. 15: Elapsed time of Mr. Scan for the sky survey dataset at 0.00015 *Eps* and 5 *MinPts*

The weak scaling behavior of the sky survey dataset resembles that of the Twitter dataset. We see similar upward trends in time as the number of nodes increase in both partitioning methods. The major reason for the increase in running time for the file system based approach is identical to that of the Twitter the dataset (file I/O). The introduction of a message passing partitioner resulted in a similar decline to those seen with the Twitter dataset. However there is still an upward trend in run times, with the major cause of this increase attributable to the GPU DBSCAN operation. Between our smallest (102 million) and largest (1.6 billion) point scales tested, we saw an increase in GPU DBSCAN run times by 2.85x (from 134 to 383 seconds). The reason for this rise in run time is due to the sparseness of the dataset resulting in the dense box algorithm not detecting any dense clusters and less than ideal partitions being generated with higher point counts due to usage of a larger *Eps* value of 0.01 for partitioning. The sky survey dataset spans hundreds of millions of *Eps* x *Eps* boxes making it impractical for a single node to quickly build the partitions. A larger *Eps* value is used to reduce the memory constraints on the head compute node of the Mr. Scan tree at a cost of creating partitions that have unneccesarily high point counts. While a larger *Eps* value (0.01) is used for partitioning, the original *Eps* value (0.00015) is used for the actual DBSCAN clustering operation and for merging results to maintain high output quality.

6. CONCLUSION

Mr. Scan uses a hybrid/hybrid parallel implementation of DBSCAN to achieve scalability. The hybrid/hybrid design used by Mr. Scan combines the MRNet tree-based distribution network with hybrid CPU-GPU nodes. Using this hybrid/hybrid design, Mr. Scan is able to successfully cluster multi-billion point datasets in a scaleable and highly accurate fashion. The three core techniques responsible for Mr. Scan's scalability are partitioning, merging, and the dense box algorithm. Partitioning in Mr. Scan allowed for the creation of independently processable partitions. Output cluster quality was maintained by the inclusion of shadow regions in each independent partition. Merging the processed results from these independent partitions allows for accurate and lightweight reconstruction of clusters. The use of shadow regions and only requiring a subset of the dataset for detecting overlapping clusters resulted in a scalable and accurate merge. The use of dense box allowed Mr. Scan to preprocess some of the heaviest computational portions of the dataset resulting in a major reduction in overall runtime.

Using the above techniques we show that Mr. Scan is capable of clustering 6.5 billion points in 7.5 minutes, which is the largest known run of DBSCAN by point and node count. On the current datasets, Mr. Scan appears to achieve maximum efficiency at around 2,000 nodes. The limit suggests that the optimum per node point count is higher than the one used for testing. This leads us to believe that as we scale up to more nodes (and substantially larger point counts), we should be able to efficiently run even larger datasets.

Acknowledgements

This work is supported in part by Department of Energy grant DE-SC0010474 and National Science Foundation Cyber Infrastructure grant OCI-1234408.

REFERENCES

2013. SDSS - Baryon Oscillation Spectroscopic Survey. (April 2013). http://www.sdss3.org/surveys/boss.php.

2013. Sloan Digital Sky Survey. (April 2013). www.sdss.org.

2013. Twitter. (April 2013). https://twitter.com https://twitter.com.

Elke Achtert, Ahmed Hettab, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2011. Spatial Outlier Detection: Data, Algorithms, Visualizations. In *Advances in Spatial and Temporal Databases*. Lecture Notes in Computer Science, Vol. 6849. Springer Berlin Heidelberg, 512–516. DOI:http://dx.doi.org/10.1007/978-3-642-22922-0_41

Dong H. Ahn, Bronis R. de Supinski, Ignacio Laguna, Gregory L. Lee, Ben Liblit, Barton P. Miller, and Martin Schulz. 2009. Scalable Temporal Order Analysis for Large Scale Debugging. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA, Article 44, 11 pages. DOI:http://dx.doi.org/10.1145/1654059.1654104

Nawab Ali, Philip H. Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert B. Ross, Lee Ward, and P. Sadayappan. 2009. Scalable I/O forwarding framework for high-performance computing systems.. In *CLUSTER* (2010-03-08). IEEE, 1–10. http://dblp.uni-trier.de/db/conf/cluster/cluster2009.html#AliCIKLLRWS09

T. Ali, S. Asghar, and N.A. Sajid. 2010. Critical Analysis of DBSCAN Variations. In *International Conference on Information and Emerging Technologies 2010 (ICIET 2010)*. Karachi, Pakistan. DOI:http://dx.doi.org/10.1109/ICIET.2010.5625720

D.C. Arnold, D.H. Ahn, B.R. de Supinski, G.L. Lee, B.P. Miller, and M. Schulz. 2007. Stack Trace Analysis for Large Scale Debugging. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. 1–10. DOI:http://dx.doi.org/10.1109/IPDPS.2007.370254

C. Böhm, R. Noll, C. Plant, and B. Wackersreuther. 2009. Density-Based Clustering using Graphics Processors. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. ACM, Hong Kong, China, Article 1646038, 10 pages. DOI:http://dx.doi.org/10.1145/1645953.1646038

Aron Culotta. 2013. Lightweight Methods to Estimate Influenza Rates and Alcohol Sales Volume from Twitter Messages. *Language Resources and Evaluation* 47, 1 (2013), 217–238.

Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE* 5, 1 (1998), 46–55.

Bi-Ru Dai and I.-Chang Lin. 2012. Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition. In *IEEE 5th International Conference of Cloud Computing (IEEE CLOUD 2012)*. Honolulu, HI, USA.

S. Davidoff and P. Wozniak. 2003. RAPTOR-scan: Identifying and Tracking Objects Through Thousands of Sky Images. In *Gamma-Ray Bursts: 30 Years of Discovery: Gamma-Ray Symposium*. Santa Fe, NM, USA.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1, Article 1327492 (Jan. 2008), 7 pages. DOI:http://dx.doi.org/10.1145/1327452.1327492

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *The Second International Conference on Knowledge Discovery and Data Mining (KDD '96)*. Portland, OR, USA.

Todd Gamblin, Bronis R. de Supinski, Martin Schulz, Robert J. Fowler, and Daniel A. Reed. 2010. Clustering Performance Data Efficiently at Massive Scales. In *ACM/SIGARCH International Conference on Supercomputing (ICS 2010)*. Epochal Tsukuba, Tsukuba, Japan.

Y. He. 2013. Personal communication. (19 March 2013).

Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. 2011. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. In *The 17th IEEE International Conference on Parallel and Distributed Systems (ICPADS '11)*. Tainan, Taiwan.

Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. 2004. DBDC: Density Based Distributed Clustering. In *Int. Conf. on Extending Database Technology (EDBT '04)*. Heraklion, Crete, Greece, 88–105.

A. Karandikar. 2010. *Clustering Short Status Messages: A topic model based approach*. Master's thesis. University of Maryland, Baltimore County.

Slava Kisilevich, Florian Mansmann, and Daniel A. Keim. 2010. P-DBSCAN: A Density Based Clustering Algorithm for Exploration and Analysis of Attractive Areas Using Collections of Geo-Tagged Photos. In *1st International Conference on Computing for Geospatial Research & Application (COM.Geo '10)*. Washington, DC, USA.

Marzena Kryszkiewicz and Piotr Lasek. 2010. TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality. In *The Seventh International Conference of Rough Sets and Current Trends in Computing (RSCTC 2010)*. Warsaw, Poland.

Marzena Kryszkiewicz and Lukasz Skonieczny. 2005. Faster Clustering with DBSCAN. In *International Conference on Intelligent Information Systems 2005: New Trends in Intelligent Information Processing and Web Mining (IIPWM 2005)*. Gdansk, Poland, 605–614.

Vasileios Lampos. 2012. On Voting Intentions Inference from Twitter Content: A Case Study on UK 2010 General Election. *ACM Computing Research Repository (CoRR)* abs/1204.0423 (2012).

Vasileios Lampos and Nello Cristianini. 2012. Nowcasting Events from the Social Web with Statistical Learning. *ACM Transactions on Intelligent Systems and Technology (ACM TIST)* 3, 4 (2012), 72.

Thomas Lansdall-Welfare, Vasileios Lampos, and Nello Cristianini. 2012. Effects of the Recession on Public Mood in the UK. In *22nd International World Wide Web Conference (WWW '12)*. Lyon, France, 1221–1226.

Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.* 22 (1996), 789–828.

R Mills, Forrest M. Hoffman, Jitendra Kumar, and William W. Hargrove. 2011. Cluster Analysis-Based Approaches for Geospatiotemporal Data Mining of Massive Data Sets for Identification of Forest Threats. *Procedia CS* 4 (2011), 1612–1621.

Frank Mueller. 1993. A library implementation of POSIX threads under UNIX. In *In Proceedings of the USENIX Conference*. 29–41.

Md. Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei keng Liao, Fredrik Manne, and Alok N. Choudhary. 2012. A New Scalable Parallel DBSCAN Algorithm using the Disjoint-Set Data Structure. In *ACM/IEEE Supercomputing Conference 2012 (SC 2012)*. Salt Lake City, UT, USA.

Md. Mostofa Ali Patwary, Nadathur Satish, Narayanan Sundaram, Fredrik Manne, Salman Habib, and Pradeep Dubey. 2014. Pardicle: Parallel Approximate Density-based Clustering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, Piscataway, NJ, USA, Article 2683655, 12 pages. DOI:http://dx.doi.org/10.1109/SC.2014.51

P. Roth, D. Arnold, and B. Miller. 2003. MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools. In *ACM/IEEE Supercomputing Conference 2003 (SC 2003)*. Phoenix, Arizona.

Martin Schulz, Dong Ahn, Andrew Bernat, Bronis R. de Supinski, Steven Y. Ko, Gregory Lee, and Barry Rountree. 2005. Scalable Dynamic Binary Instrumentation for Blue Gene/L. *SIGARCH Comput. Archit. News* 33, 5, Article 1127581 (Dec. 2005), 6 pages. DOI:http://dx.doi.org/10.1145/1127577.1127581

S. Sonntag, C. T. Paredes, J. Roth, and H.-R. Trebin. 2011. Molecular Dynamics Simulations of Cluster Distribution from Femtosecond Laser Ablation in Aluminum. *Applied Physics A* 104, 2 (2011), 559–565.

Telegraph. 2013. Library of Congress Is Archiving All Of America's Tweets. (January 2013). http://www.businessinsider.com/library-of-congress-is-archiving-all-of-americas-tweets-2013-1.

Benjamin Welton and Barton P. Miller. 2014. The Anatomy of Mr. Scan: A Dissection of Performance of an Extreme Scale GPU-based Clustering Algorithm. In *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '14)*. IEEE Press, Piscataway, NJ, USA, Article 2691150, 7 pages. DOI: http://dx.doi.org/10.1109/ScalA.2014.10

Benjamin Welton, Evan Samanas, and Barton P. Miller. 2013. Mr. Scan: Extreme Scale Density-based Clustering Using a Tree-based Network of GPGPU Nodes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 84, 11 pages. DOI: http://dx.doi.org/10.1145/2503210.2503262

Robert E. Wilson, Samuel D. Gosling, and Lindsay T. Graham. 2012. A Review of Facebook Research in the Social Sciences. *Perspectives on Psychological Science* 7, 3 (2012), 203–220. DOI: http://dx.doi.org/10.1177/1745691612442904

Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. 1999. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Mining and Knowledge Discovery* 3, 3 (1999), 263–290.