The Anatomy of Mr. Scan: A Dissection of Performance of an Extreme Scale GPU-Based Clustering Algorithm

Benjamin Welton and Barton P. Miller
Computer Sciences Department
University of Wisconsin
Madison, WI 53706
{welton,bart}@cs.wisc.edu

***Abstract*—The emergence of leadership class systems with GPU-equipped nodes has the potential to vastly increase the performance of existing distributed applications. However, the inclusion of GPU computation into existing extreme scale distributed applications can reveal scalability issues that were absent in the CPU version. The issues exposed in scaling by a GPU can become limiting factors to overall application performance. We developed an extreme scale GPU-based application to perform data clustering on multi-billion point datasets. In this application, called Mr. Scan, we ran into several of these performance limiting issues. Through the use of complete end-to-end benchmarking of Mr. Scan (measuring time from reading and distribution to final output), we were able to identify three major sources of real world performance issues: data distribution, GPU load balancing, and system specific issues such as start-up time. These issues comprised a vast majority of the run time of Mr. Scan. Data distribution alone accounted for 68% of the total run time of Mr. Scan when processing 6.5 billion points on Cray Titan at 8192 nodes. With improvements in these areas, we have been able able to cut total run time of Mr. Scan from 17.5 minutes to 8.3 minutes when clustering 6.5 billion points.**

Keywords*—*Distributed Systems, GPU Data Clustering, DBSCAN, Performance Analysis

I. Introduction

Discussions of heterogeneous systems have come in two flavors: (1) message passing combined with shared memory (often described as MPI+OpenMP) and (2) CPU combined with GPU. In reality, we are now confronted with both of these flavors on leadership class systems (and even on clusters)

Our experience with scalable computing [9], [2], [10], [7] and, most recently, with extreme scale cluster algorithms [11] has shown us that these two flavors interact and cannot be treated separately. We call this combined model the *distributed CPU/ GPU model*.

In this combined distributed CPU/GPU model, the addition of GPUs affects the scalability characteristics of the algorithm as a whole. Suboptimal design choices in other parts of the model that did not limit scalability in a CPU-only implementation become limiting factors with the addition of a GPU. In an extreme scale clustering algorithm we developed (called Mr. Scan [11]), we saw the impacts that these issues can have on end-to-end application performance when running on leadership class systems at scale. Data distribution, GPU load balancing, and application start-up time were major limiting factors to the scale up of Mr. Scan. In this paper we investigate the scalability issues brought up by the inclusion of GPUs in Mr. Scan.

Mr. Scan is a distributed clustering algorithm that uses GPUs for clustering multi-billion point datasets. 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. Mr. Scan is our implementation of the DBSCAN (Density Based Spatial Clustering of Applications with Noise) clustering algorithm [6]. Mr. Scan uses the MRNet tree based overlay network [9] to organize processes into a multi-level tree with an arbitrary topology. In the 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 the first implementation of DBSCAN that can scale up to multi-billion data point datasets and the first distributed DBSCAN algorithm that incorporates the use of GPUs.

With Mr. Scan, we focused on end-to-end performance. End-to-end performance is the total run time of the application from reading and distribution of input data to writing the final output. The use of end-to-end performance as a metric gives context to how the application will perform in a real world setting. End-to-end benchmarking allows us the opportunity to address scalability issues that would come up in actual use but do not show up in traditional benchmarks. In a distributed application that uses GPU-based computation, end-to-end benchmarking becomes a more important metric because the increased performance of the algorithm stresses other portions of the distributed application. Data distribution, communication, load balancing, and system costs like start-up time become more dominate factors in application performance due to reduced processing time by use of a GPU. Each of these issues are major contributing factors of run time for Mr. Scan. All combined, they composed 86% of the total run time of Mr. Scan with only 14% being spent on actually processing data.

Data distribution had the largest impact on scalability of Mr. Scan. Data distribution is the act of getting the data to the nodes for computation. In our original benchmarks of Mr. Scan, data distribution comprised 68% of the total run time of the application (11.8 minutes out of a total run time of 17.8 minutes at 8192 nodes). A vast majority of the time spent

on data distribution was writing intermediary files that would be created by the data partitioner and then loaded by each compute node. In a CPU-based implementation, using intermediary files to pass information to compute nodes would have been an acceptable (though inefficient) strategy for loading data into the node for computation due to the much longer computation time of DBSCAN on the CPU. However, with the increased performance of the GPU, using intermediary files becomes a limiting factor to scaling. We reduced the cost of data distribution significantly by directly passing data to the compute nodes via a message passing interface.

Load balancing in general is a concern of any distributed algorithm. Since a distributed application is only as fast as its slowest component, having a good balance between nodes is essential to high performance in a distributed application. With the introduction of a GPU-based DBSCAN implementation, obtaining this balance became difficult because the size of the input data did not directly correlate with compute time on the GPU. Identically sized input data could have more than a 10x difference in run time due to the effect of input data density on run-time complexity; denser datasets run closer to the worst-case complexity of the DBSCAN algorithm. Achieving good load balancing required that we take density into account when partitioning data, as well as find ways to reduce the run-time cost of clustering dense data regions. Load balancing of Mr. Scan was achieved by using the GPU itself to identify dense regions of data and to cluster them in a more efficient manner.

Machine and system-specific issues have a measurable impact on Mr. Scan's overall performance. The machine issues are related to start-up and shutdown time. The cost of these operations was surprisingly high at large scale. At 8192 nodes, start-up of the application took more time than the entire DBSCAN GPU computation phase. Unfortunately, this issue is not one that can be resolved by a user since it requires changes to the launching and shutdown process of the system environment. However, techniques may be made available in the future to reduce the cost of start-up. While most application benchmarks do not include this time (and thus do not find it a scalability concern), we included these numbers in our results.

Debugging was also a challenge with Mr. Scan due to the inclusion of GPUs. The non-deterministic ordering of thread block execution in the GPU increased the difficulty of verifying the correctness of results after a change to the Mr. Scan code base. Different thread block execution ordering on a node would produce slightly different, yet algorithmically correct, output. Validating results required the use of specially written tools to check Mr. Scan output for correctness, which was costly both in terms of programmer and computation time.

In Section II, we give a description of the DBSCAN algorithm itself. In Section III, we describe the Mr. Scan algorithm in more detail. We then discuss the scalability challenges of Mr. Scan in Section IV. Finally, we wrap up with our concluding thoughts.

II. The DBSCAN algorithm

The DBSCAN algorithm generates clusters of data points based on density. The use of density as the metric allows for the detection of irregularly shaped clusters without prior knowledge of the number of clusters in the dataset. DBSCAN's notion of density comes from its two parameters, *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.

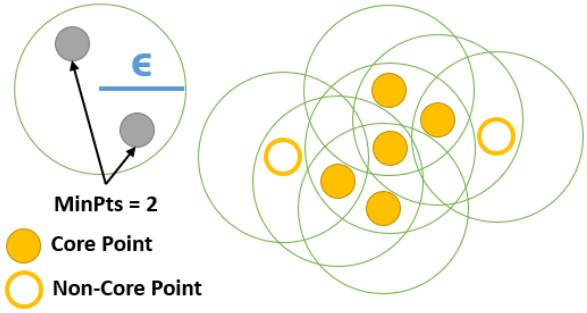

Fig. 1: DBSCAN clustering example showing classification of *core* and *non-core* points in a dataset.

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. The 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 that 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., an R*-tree or KD-tree).

III. Mr. Scan Algorithm

Mr. Scan is a parallel implementation of the DBSCAN algorithm with two phases, the partitioning phase and the clustering phase. The partitioning phase distributes a raw unordered input file containing point data to leaf nodes for the clustering phase. The clustering phase performs the DBSCAN clustering operation and generates an output file containing the final clustered 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.

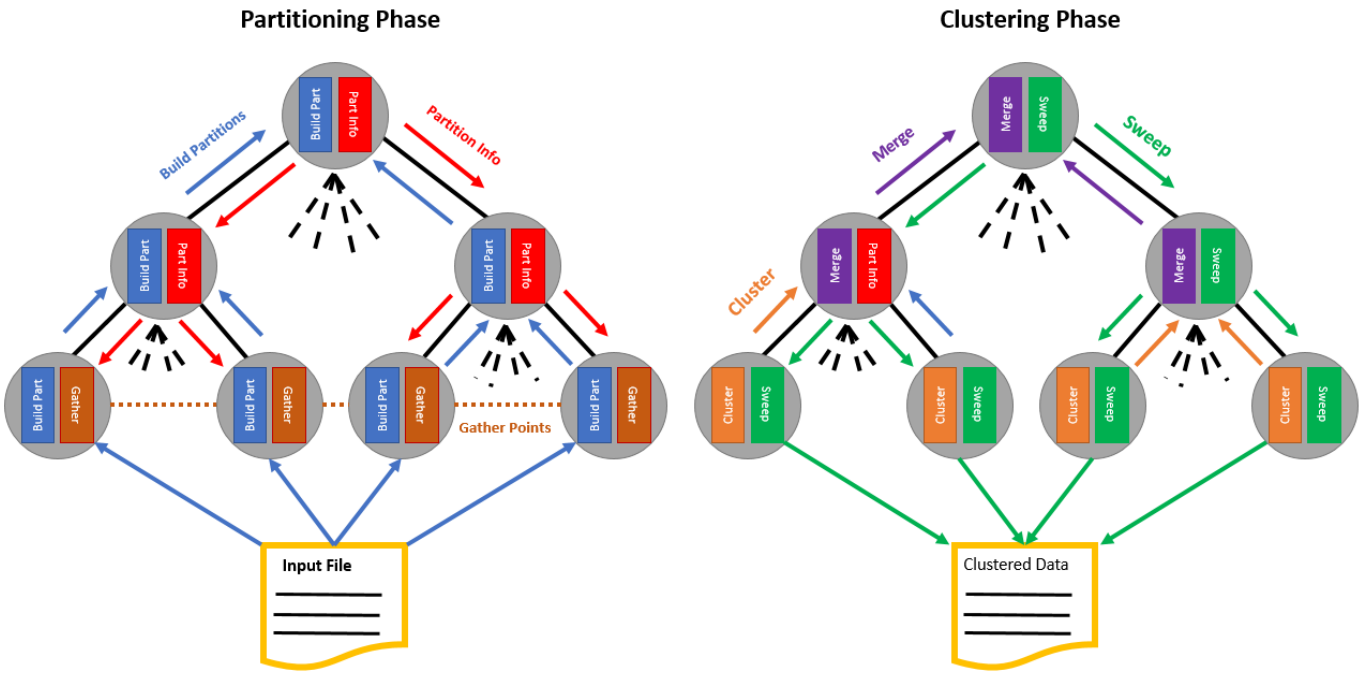

Fig. 2: The Mr. Scan algorithm

A. Partitioning Phase

In the partition phase, the input file is read by a partitioner that creates one partition per clustering process (one partition per leaf node). The input file can contain billions of points and can reach sizes up to 300 GB, so the partitioner is distributed using a hybrid MRNet/message passing model to parallelize this step. Each leaf node process starts by reading a unique section of the input file. Input points read from the file are then used to construct a *Eps* $\times$ *Eps* grid on each leaf node. The point counts of each grid cell are then sent by the leaf nodes up to the root of the tree. At the root the grid cells are then used to generate the partitions for DBSCAN. Each partition contains a set of *Eps*-grid cells. The partition will also contain a set of *Eps*-grid cells that overlap with adjacent partitions called a *shadow region*. The shadow region allows for DBSCAN to cluster each partition independently from one another. The grid cells contained by each partition are then sent back to the leaf nodes. The leaf nodes perform a gather operation (all-to-all operation) to move the locally stored points contained in each partition to the leaf node that will process that DBSCAN partition. Each leaf node ends up with a single complete partition of data.

B. Clustering Phase

The clustering phase consists of three steps: DBSCAN clustering, result merging, and a sweep step to write the output. The DBSCAN clustering phase is performed on the leaf nodes. Each leaf node performs the DBSCAN clustering independent of one another. Mr. Scan generates a set of representative points for each cluster detected on the leaf nodes. The representative points are used to reduce the amount of data needed to perform the merge operation.

The merge step is used to combine clusters found on different leaf nodes which contain overlapping core points. The merge phase takes place in the non-leaf nodes of the tree. The merge phase is needed because of the shadow region present on leaf nodes. A cluster found on a leaf node that spans into the shadow regions has a chance of being an extension of a cluster found on a different leaf node. The merge algorithm uses the representative points found during the clustering operation to detect overlaps between regions. The merge is performed at all non-leaf node levels of the tree.

Finally, the sweep step is performed to generate the output file. The root node gives each cluster generated by the last merge operation a unique identifier. The root then passes the merged clusters back down the tree to the leaf nodes for writing the final output.

IV. Scaling Challenges of Mr. Scan

Mr. Scans scaling limitations come from components outside of the main algorithm. Data distribution, load balancing, and start-up time were the significant contributors to Mr. Scan's total run time, making the application show poor weak scaling properties. While these issues are not specific to Mr. Scan and are common among distributed applications, the use of GPU processing increases the impact that each of these has on application scalability. Without the use of end-to-end benchmarking, performance issues outside of the main algorithm component would not have been known. Figure 3 illustrates why this type of comprehensive benchmark is necessary. The algorithm component of Mr. Scan shows decent weak scaling properties; however, the total run time shows poor overall weak scaling. While the poor weak scaling numbers have obvious implications for real world usage, another issue is comparing results between implementations of the same algorithm. It

is not uncommon for other papers that describe a parallel implementation of DBSCAN to only show the equivalent of the algorithm component of Mr. Scan in their results. They ignore the partitioning and pre-processing costs required before running DBSCAN. Since these operations are implementation specific, not having any idea what they cost to perform makes comparisons between algorithms difficult.

In our end-to-end benchmarks of Mr. Scan, the importance of the performance outside of the algorithm component proved to be critical. In fact, the total run time of the algorithmic component at our largest scale (8192 nodes with 6.5 billion points from a dataset composed of geo-located tweets obtained from Twitter [1]) was not the largest contributor to run time. Partitioning and start-up costs accounted for more overall time than DBSCAN itself. Figure 4 shows the breakdown of run times for the components of the original implementation of Mr. Scan.

While we have not been able to fully eliminate the scaling issues brought up by both the partitioning and application start-up phases of Mr. Scan, we have been able to make some improvements. In Figure 5, we show a comparison between the original Mr. Scan implementation and our improved implementation (running on both the Widow and Atlas file systems on Cray Titan). We show a breakdown of performance of the improved Mr. Scan in Figures 6 and 7. The majority of the improvement we see is in data distribution time. We discuss data distribution is a more detail in Section IV-A. We also go over the other system specific costs of Mr. Scan in Section IV-C.

A. Data distribution

The original Mr. Scan implementation used a partitioner that was separate from the DBSCAN phase. The partitioning program acted completely independently from the actual clustering as a seperate distributed application. Performing the partitioning was done by reading the complete dataset, building the partitions, and writing the complete set of partitions out to disk which would then be read by the DBSCAN processes. At 6.5 billion points the read operation by the partitioner took 229 seconds and the complete write of the partitioned data took 489 seconds. The write operation alone took 39% of Mr. Scan's total run time to perform (489 seconds out of total run time of 1226 seconds).

We eliminate writing the partitioned data to the file system by combining both the partitioner and the DBSCAN process into a single application. Instead of writing the partitioned data out to disk we pass the partitions directly to the processing nodes. The total time to get partitioned data to the DBSCAN processes has been reduced from 489 seconds down to 15 seconds at 6.5 billion points. In figure 8 we show the total run time of the partitioning phase of Mr. Scan with message passing. In the runs shown, 90% of the run time of the partitioner is spent on the initial dataset read from the file system.

B. GPU load balancing

Load balancing DBSCAN is a challenge due to the effect of density on algorithm performance. When DBSCAN processes a point, it searches for neighbor points within *Eps* distance. The performance of the search operation is tied to the spatial index used for neighbor lookup. Spatial indexes in common use by DBSCAN perform at their worst in regions that are exceptionally dense. Slow performance in dense regions affects the spatial index used by Mr. Scan. Mr. Scan's GPU algorithm, which is a slightly modified version of the CUDA-DClust DBSCAN implementation [5], performed at $O(n^2)$ in extremely dense regions due to the spatial index used.

The spatial index in use by Mr. Scan is a modified KD-Tree [4] with fixed height of two, one for each dimension of the data. The major difference between the KD-Tree used by Mr. Scan and a standard KD-Tree is that nodes represent regions of data instead of single points. Leaf nodes in the KD-Tree point to data within a (X,Y) value range while internal nodes point to leaf nodes containing data in distinct non-overlapping X value ranges. The nodes are stored in the tree in sorted order relative to their parent; the left-most child node contains the range with the lowest minimum value and the right-most child node contains the range with the largest maximum value. Each leaf node of the KD-Tree contains an offset that acts as a pointer into a array where the point data is stored. In addition, each leaf node of the KD-Tree also contains the number of points inside the leaf node's range. The leaf nodes are given

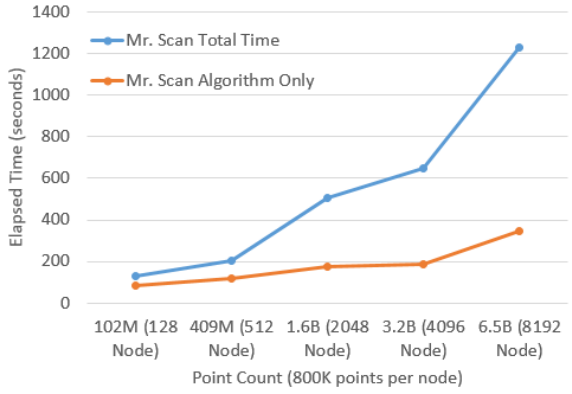

Fig. 3: Mr. Scan algorithm only and total run time on a Twitter dataset.

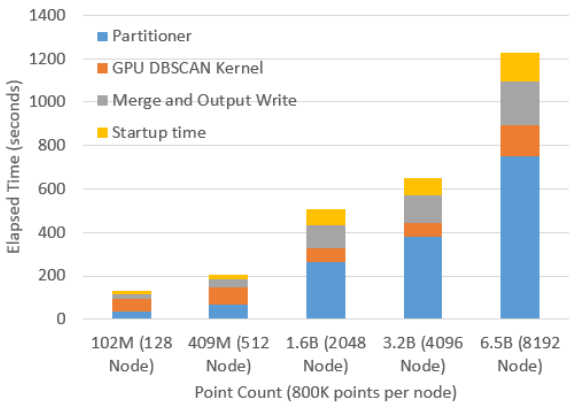

Fig. 4: A breakdown of the run times of various components of the original implementation of Mr. Scan.

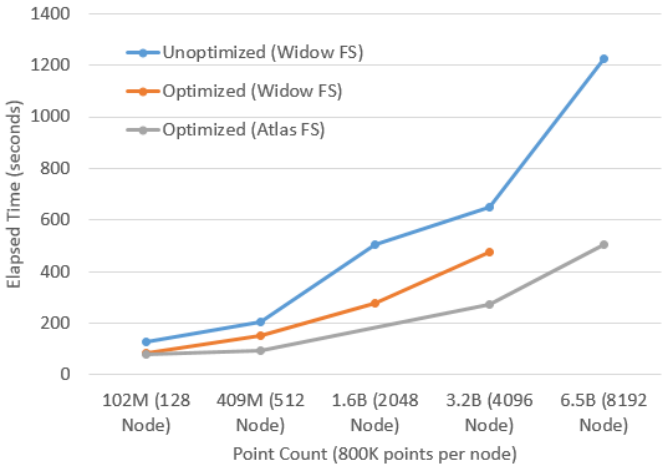

Fig. 5: Total run time for Mr. Scan on a Twitter dataset.

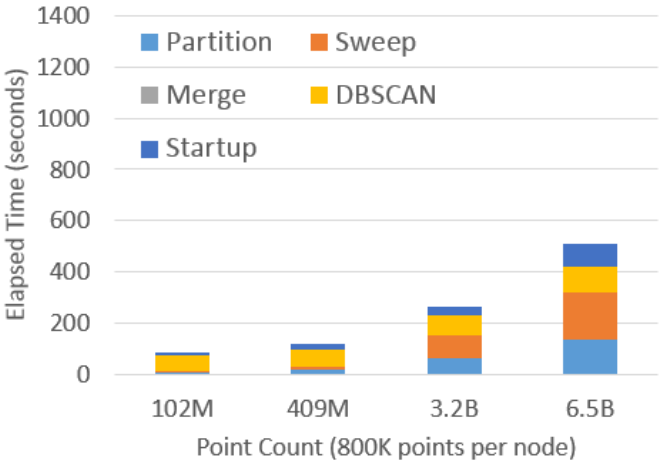

Fig. 7: Breakdown of the Mr. Scan using message passing on the Atlas file system

offsets in a sequential fashion, with the left-most leaf node of the tree at offset 0 and the right-most at offset N (the total number of points).

The purpose of the modified KD-Tree in Mr. Scan is to reduce the number of possible neighbor points for the point that is being processed. Since each possible neighbor point has to be compared to the point being processed to determine if they are within *Eps* distance of each other, reducing the number of possible neighbor points can improve performance. The KD-Tree reduces the number of possible neighbor points by eliminating points that cannot be neighbors. The neighbor search is done by performing two lookups in the KD-Tree. The two lookups find both the starting node and the ending node for neighbor comparison. The starting node is the leaf node of the KD-Tree furthest left in the tree that contains a possible neighbor point. The starting node's offset becomes the offset in the point array where DBSCAN begins search for neighbor points. The ending node is the furthest node right (maximum X and Y value range) in the tree that contains a possible neighbor point generating the ending offset in the point array. The point being processed is then compared to every point

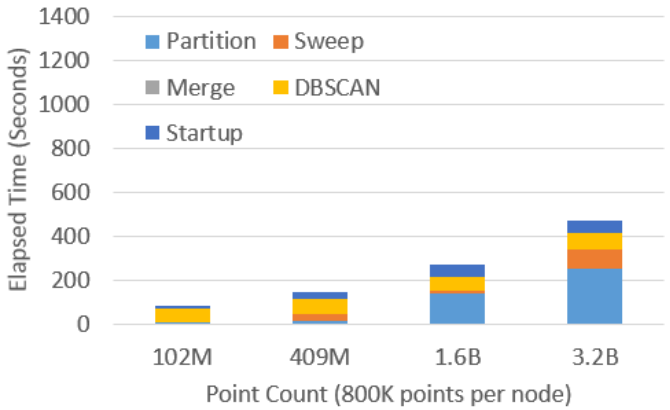

Fig. 6: Breakdown of the Mr. Scan using message passing on the Widow file system.

between the starting and ending offset. In less dense regions, the lookup operation will produce starting and ending offsets that are closer to one another. In extremely dense regions, the starting and ending offsets can span significant portions of the dataset (in some cases all of the dataset).

Density makes load balancing complicated because the Twitter dataset does not have uniform density, leading to some of the GPU nodes performing more than an order of magnitude slower (having to perform comparison operations on significantly larger portions of the dataset). Even though we could detect dense regions in the partitioner, the dense regions detected could not be broken apart because they would still need to be processed by a single GPU due to proper point classification (*core* and *non-core* point classification) of DBSCAN needing access to all neighbor points.

Our solution to load balancing was to focus on reducing the performance penalty of these dense regions. Load balancing was achieved by using an algorithm we developed called dense box, which detects these dense regions and groups them as a cluster prior to running DBSCAN. Dense box detects these regions by looking at the leaf nodes of the modified KD-Tree spatial index, looking for regions with point counts greater than *MinPts* in size with a area of less then *Eps*. When these regions are detected, the points inside of these regions are marked as being members of a the same cluster. When these points are processed by DBSCAN, no neighbor lookup is performed. The effect of dense box on run times can be seen at relatively small scales. At 512 nodes (409 million points), the slowest node in processing DBSCAN with dense box was 94 seconds, while the slowest node without dense box on the same dataset was 743 seconds. The time disparity gets progressively worse at larger scales; at 1024 nodes the difference is around 26 minutes.

C. System challenges

One of the more surprising elements uncovered in our testing of Mr. Scan was the effect that the Cray application launcher had on performance. Application start-up consumed

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 does 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. We plan to evaluate the facility once it is available to us. The second problems 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.

D. Debugging Challenges of Mr. Scan

Non-deterministic execution order of thread blocks in the GPU made debugging Mr. Scan challenging. Different thread block execution orders can result in different, yet algorithmically correct, output. The variation in output produced by different runs makes it difficult to determine if a code change to Mr. Scan maintained result quality. There are two properties of DBSCAN that cause the variation in output: by different runs is cluster identifier selection and cluster selection of non-core border points (non-core points within an Eps radius of two or more clusters). Naive methods of validating output (such as checksumming) were not usable due to different output being produced from each run. We had to create special validation tools to help determine the quality of the result. Developing, maintaining, and executing Mr. Scans validation tools is a time consuming process. Having to develop validation tools specifically for Mr. Scan is less than ideal and we would have preferred a more generic method of ensuring correctness.

V. Unresolved Issues

There are still a few remaining performance issues that have yet to be addressed. File I/O continues to be an issue even though we have resolved the distribution problem. While the performance of the I/O sections (initial file read and writing output) of Mr. Scan is acceptable, the performance is slower than we anticipated. The simple strategy that we use for reading and writing data (having every node read/write a separate section of a file in a contiguous fashion) does not seem to be well-suited for high node counts. Resolution of the I/O issue may require finding the correct number of nodes reading/writing for high efficiency or using dedicated I/O forwarders to reduce the number of nodes interacting with the file system.

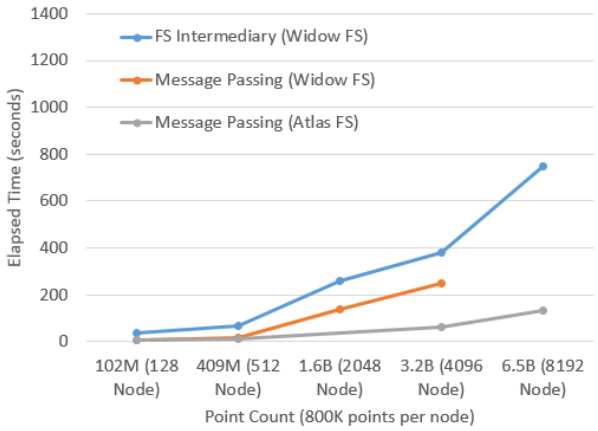

Fig. 8: Partitioner times with the original file system based partitioner and message passing partitioner.

An issue that we have not addressed is how to recover from failures in Mr. Scan. Since we are focusing on real world performance, we consider failure recovery a key unresolved issue. While there are methods for failure recovery discussed in a variety of other distributed systems, including the distribution framework used for Mr. Scan [3], there has been little work on GPU-specific failure recovery at extreme scales. While generic failure recovery methods will work for Mr. Scan, these generic systems do not take advantage of the finer decomposition at which the computation is defined for the GPU (by kernel executions and thread groups). The natural decomposition provided by the GPU may allow for an opportunity to accelerate failure recovery (for example by spreading the computation from the failed GPU node across all remaining nodes).

VI. Conclusion

GPUs have the ability to vastly increase the performance of distributed applications. Getting full real world benefit from the inclusion of GPUs requires that application developers take into account the effect that the inclusion of GPU processing in a distributed application will have on the other portions of the application. In the case of Mr. Scan, we used end-to-end benchmarking to successfully identify and solve several problem areas where performance was impacting scalability. Data distribution, which accounted for 68% of run time in our unoptimized version of Mr. Scan, was reduced to only 26% of total run time. The use of message passing instead of writing intermediary files to the file system was responsible for the reduction in data distribution time. GPU load balancing was also improved by reduced by increasing the efficiency of processing extremely large regions of data. While resolution of performance issues found by end-to-end testing has led to performance gains there are still some unanswered problems. File I/O and failure recovery remain two unsolved issues with real world performance implications.

References

[1] Twitter, April 2013. https://twitter.com.

[2] D. Arnold, D. Ahn, B. De Supinski, G. Lee, B. Miller, and M. Schulz. Stack trace analysis for large scale debugging. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, 2007.

[3] D. Arnold and B. Miller. Scalable failure recovery for high-performance data aggregation. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9), 1975.

[5] C. Böhm, R. Noll, C. Plant, and B. Wackersreuther. Density-based clustering using graphics processors. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 661–670, 2009.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. 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, Aug. 1996.

[7] E. R. Jacobson, M. J. Brim, and B. P. Miller. A lightweight library for building scalable tools. In *Applied Parallel and Scientific Computing*, volume 7134 of *Lecture Notes in Computer Science*, pages 419–429. 2012.

[8] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing. The application level placement scheduler. 2006.

[9] P. Roth, D. Arnold, and B. Miller. Mrnet: A software-based multicast/reduction network for scalable tools. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 21–21, 2003.

[10] P. C. Roth and B. P. Miller. On-line automated performance diagnosis on thousands of processes. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '06, pages 69–80, 2006.

[11] B. Welton, E. Samanas, and B. P. Miller. Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 84:1–84:11, 2013.