DiscFinder: A Data-Intensive Scalable Cluster Finder for Astrophysics

Bin Fu binf@cs.cmu.edu
Kai Ren kair@cs.cmu.edu
Julio López jclopez@cs.cmu.edu
Eugene Fink e.fink@cs.cmu.edu
Garth Gibson garth@cs.cmu.edu
School of Computer Science, Carnegie Mellon University

ABSTRACT
DiscFinder is a scalable approach for identifying large-scale astronomical structures, such as galaxy clusters, in massive observation and simulation astrophysics datasets. It is designed to operate on datasets with tens of billions of astronomical objects, even in the case when the dataset is much larger than the aggregate memory of compute cluster used for the processing.

1. INTRODUCTION
Today, discoveries in data-driven sciences, such as astrophysics, seismology and bio-informatics, are often enabled by analysis of massive datasets. In particular, the advent of digital astronomical surveys, beginning with the Sloan Digital Sky Survey[2], has dramatically increased the size of astronomical data. Similarly, modern cosmological simulations, such as BHCosmo[4] and Coyote Universe[9], produce datasets with billions of objects, and multiple terabytes in size.

Companies that provide Internet services, such as search and social networks, deal with datasets of similar size. Frameworks, such as Hadoop and Dryad, are commonly used for data-intensive applications on Internet-scale data, such as text analysis and indexing of web pages. *Can we leverage these frameworks for science analytics?* In this work, we want to understand the requirements for enabling science analytics using these systems.

We present our work on DiscFinder, which is a data-intensive approach for finding clusters of astronomical objects, implemented on top of Hadoop. Although, it is specific to astrophysics, we expect that similar principles are applicable to clustering problems in other sciences. The main benefits of this approach, and future science analytics applications implemented using these frameworks, are simplicity, fast development and scalability.

This work was sponsored in part by grants from Google, The Moore Foundation, National Science Foundation Open Cloud Consortium, The Petascale Data Storage Institute (PDSI), The McWilliams Center for Cosmology and the support of the companies of the Parallel Data Laboratory Consortium (PDL)

HPDC 2010 Chicago, IL USA

The described technique scales to datasets with tens of billions of astronomical objects. It leverages sequential algorithms to cluster relatively small subsets of input objects. The main idea is to partition the input dataset into regions, then execute a sequential clustering procedure for each partition, and finally merge the results, joining the clusters across partitions. The developed procedure relies on Hadoop for splitting the input data, coordinating the execution of tasks and handling task failures.

2. MOTIVATING APPLICATION
To understand the evolution of the universe, astrophysicists analyze large-scale astronomical structures[17]. Finding the clusters of objects that make up large-scale structures is the first step towards this analysis. In astrophysics, *group finding* refers to a family of clustering algorithms. Their input is a set of astronomical objects, such as stars or galaxies. We refer to those objects as particles or points. A group finder identifies subsets of particles that form larger structures, such as galaxy halos, groups and clusters. We use the generic term *groups*.

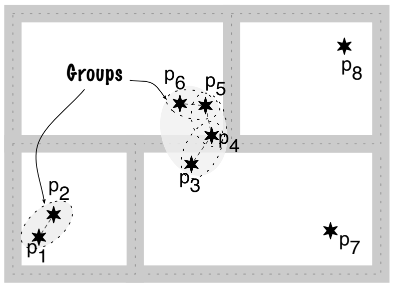


Figure 1: Friends of Friends (FOF) clustering.

Huchra and Geller have developed a basic group-finding algorithm, called *Friends-of-Friends* (FOF) [10]. Although it analyzes only distances between astronomical objects, and disregards their masses, velocities, and other relevant factors, it has been proved effective in finding galaxy clusters and has become one of the standard astrophysics tools.

The FOF algorithm uses two parameters: a *linking length* (τ) and a *minimum group size* (`minGz`). Its input is a set of particles, where each particle is a tuple $\langle \texttt{pid}_i, x_i, y_i, z_i \rangle$; `pid` is the particle identifier and x_i, y_i, z_i are its coordinates. Two particles are considered "friends" if the distance between them is at most τ. The friendships between particles define an undirected graph, and its con-

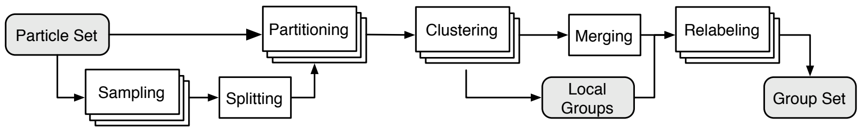


Figure 2: DiscFinder Pipeline. The DiscFinder computation flowgraph comprises the following stages: 1. Sampling (dist), 2. Splitting (seq), 3. Partitioning (dist), 4. Clustering (dist), 5. Merging (seq), and 6 Relabeling (dist).

nected components correspond to groups of particles (see Figure 1). The output of FOF comprises the groups that contain at least `minGz` particles.

Researchers have developed several sequential versions of FOF such as the UW-FOF from the "N-body shop" at the University of Washington[16]. Parallel group finders, such as pHOP[12], Ntropy[7], Halo World[14], Amiga Halo Finder (AHF)[8] and are implemented using the Message Passing Interface (MPI)[13] and designed for parallel distributed memory machines with a fast low-latency interconnect between hosts. All these approaches require the complete dataset to fit in memory. Their reported performance excludes the time of loading the data. Recently, Kwon et al. developed an approach atop Dryad[11]. It shares similarities with our work. They have shown results for clustering datasets with up to 1 billion particles using eight nodes. We have developed an alternative data-intensive group finding approach, named DiscFinder, which allows processing datasets that are much larger than the memory of available compute resources[5].

3. DiscFinder

DiscFinder is a distributed out-of-core group finder. The basic idea is to take a large *unordered* particle set, split it into multiple spatial regions, find groups in each region using a sequential group finder and merge the results from the individual regions. DiscFinder is implemented as a series of MapReduce jobs using Hadoop[1, 3].

Hadoop provides a mechanism for applying a user-defined procedure (*map function*) over a large dataset, then grouping the results by a key produced by the map function, and finally applying another user-defined procedure (*reduce function*) to the resulting groups. The framework partitions the input data based on a user-defined criterion, coordinates the distributed execution of the map and reduce functions, and handles the recovery of failed tasks. Whenever possible, the framework co-locates the computation with the input data.

The DiscFinder pipeline stages are shown in Figure 2. They are: Sampling, Splitting, Partitioning, Clustering, Merging, and Relabeling. The input is the set of particles and the output is the set of groups. Sampling is a parallel stage that reads a small sample of the input dataset. The Splitting stage builds a *kd*-tree using the sample data, which is much smaller than input dataset. The *kd*-tree is used to construct the spatial partition for the domain (split boxes). Sampling and Partitioning are performed as a MapReduce job.

Partitioning and Clustering are implemented as a separate MapReduce job shown in Figure 3. The Partitioning stage uses the split boxes to determine to which partition a particle belongs. Each partition is processed independently and there is no explicit communication across tasks that process different partitions. DiscFinder uses a *shell-based partitioning* scheme that enables the independent and asynchronous processing of each partition by decoupling the clustering computation inside a partition from the resolution of groups that span across partitions. This has a small increase in the memory requirement for each partition. The partitions are extended by $\tau/2$ on each side, where τ is the linking length parameter. The end result is that a partition has a shell around it that contains points shared with adjacent partitions. This approach works under the assumption that the size of the shell is much smaller than the size of the partition. In practice, commonly used values for τ are relatively small compared to the partition edge length.

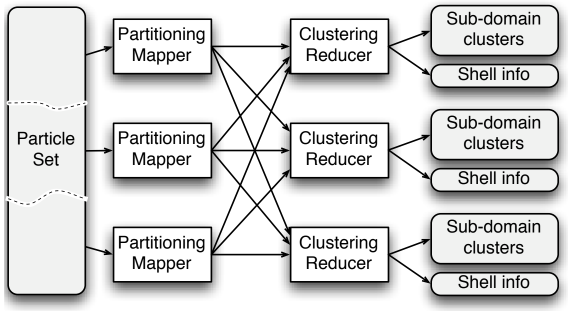


Figure 3: DiscFinder Partition and Clustering stages. This is the central MapReduce job in the DiscFinder pipeline

Clustering is a distributed stage where each task independently processes a spatial partition of the data by locally executing a sequential clustering implementations, such as UW's FOF or aFOF[16]. The tasks generate $\langle$`pid`, `lGroupId`$\rangle$ pairs that identify the group of each particle. The group identifier (`lGroupId`) is local to each partition. The pairs are split into two separate sets: one for the shell of the partition and one for the interior region. The purpose of the Merging stage is to consolidate groups that span multiple partitions by using the group membership for particles in the shells. This is achieved using a *Union-Find* algorithm to merge subgroups that have common points[6]. Union-Find has nearly linear computational complexity and since it is only applied to a small subset of the dataset, it accounts for a small portion of the total running time. Relabeling is a parallel stage that performs a single pass over the output of the Clustering phase and relabels the particle membership according to the set of global group identifiers produced by the Merging step.

4. EVALUATION

The goal of our evaluation is to test whether DiscFinder is a feasible approach for clustering massive particle datasets from astrophysics, and indirectly whether similar approaches can be used for other large-scale analytics in science. We want to measure and understand the overheads introduced by the DiscFinder algorithms and the Hadoop framework, and thus find ways to improve both the application and the framework. We conducted a set of scalability and performance characterization experiments.

Name	Particle count	Snap size	Snap count	Total size
BHCosmo	20 M	1GB	22	22 GB
Coyote Univ.	1.1 B	32GB	20	640 GB
DMKraken	14.7 B	0.5TB	28	14 TB

Figure 4: Cosmology simulation datasets

Datasets. In our evaluation we used snapshots (time slices) from the three different cosmology simulation datasets shown in Figure 4: BHCosmo[4], Coyote Universe[9] and DMKraken from our collaborators at the CMU McWilliams Center for Cosmology. These datasets are stored using the GADGET-2 format[15].

Experimental Setup. We carried out the experiments in a data-intensive compute facility that we recently built, named the DISC / Cloud cluster. Each compute node has eight 2.8GHz CPU cores in two quad-core processors, 16 GB of memory and four 1 TB SATA disks. The nodes are connected by a 10 GigE network using Arista switches and QLogic adapters at the hosts. The nominal bi-section bandwidth for the cluster is 60 Gbps. The cluster runs Linux (2.6.31 kernel), Hadoop (0.19.1), and Java (1.6). The compute nodes serve both as HDFS storage servers and worker nodes for the MapReduce layer. Hadoop is configured to run a maximum of 8 simultaneous tasks per node: 4 mappers and 4 reducers.

Scalability Experiments. With these experiments we want to answer whether DiscFinder can be used to cluster datasets that are much larger than the aggregate memory of the available compute resources, and how its running time changes as we vary the available resources. Similar to the evaluations of compute-intensive applications, we performed strong and weak scaling experiments. The results reported below are the average of three runs with system caches flushed between runs. Some of the running times include partial task failures that were recovered by the framework and allowed the job to complete successfully.

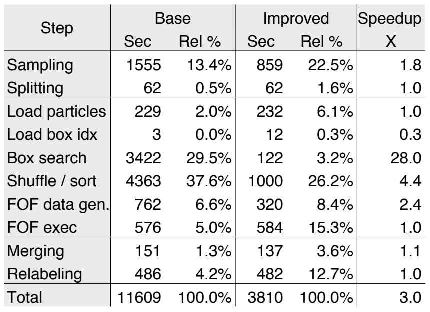
In the weak scaling experiments, the work per host is kept constant and the total work grows proportional as compute resources are added. In the strong scaling experiments, the total work is kept constant and the work per host changes inversely proportional to the number of compute nodes. We varied the number of compute hosts from 1 to 32. The same nodes were used for HDFS as well. The results are shown in figures 5 and 6. The X axis in these figures is the cluster size (number of worker nodes) in log scale.

Figure 5 shows the strong scalability of the DiscFinder approach. The Y axis is the running time in log scale. The curves correspond to different dataset sizes of 14.7, 1 and 0.5 billion particles. Notice that the 14.7 billion dataset is larger than the memory available for the experiments (16 and 32 nodes). The same applies for several scenarios in the cases for 1 and 0.5 billion particles. Linear scalability corresponds to a straight line with a slope of -1, indicating the running time decreases proportional to the number of nodes. The DiscFinder running time is sub-linear. With a small number of nodes, the running time suffers because each node performs too much work, and with a larger number of nodes each node performs too little work and the framework overhead dominates.

In weak scaling graph (Figure 6), the Y-axis is the elapsed running time in seconds (linear scale). The curves correspond to different problem sizes of 64 (top, solid line, circles) and 32 (bottom, dashed line, squares) million particles per node.

The best running time to compute nodes ratio has a sweet spot around 4 to 8 nodes for datasets smaller than 1 billion particles. We expect non-negligible overheads, introduced by the framework and the approach, due to the incurred I/O, data movement and non-linear operations such as the shuffle/sort in the job that performs the partitioning and clustering stages. The running time in all cases exhibits a non-linear trend, especially for larger number of nodes where the framework overheads dominate the running time. Even with all the aforementioned overheads and task failures, it was possible to process and cluster particle datasets that were much larger than the available memory.

Performance Characterization. We are interested in gaining insight about the DiscFinder running time. We performed additional experiments to break down the total running time into the time spent in each stage of the pipeline. For this set of experiments we used 32 nodes and focused on the largest (14.7 billion particles) dataset. The running time breakdown is shown in Figure 7. The rows corresponds to stages in the pipeline. The Partitioning and Clustering stage are broken down further into the following steps: Load particles, Load box index, Box search, Shuffle/sort, FOF data generation, and FOF execution. Columns 2 and 3 show the absolute (in seconds) and relative time (percentage of total time) for the unmodified implementation of the pipeline. The Sampling, Box search and Hadoop Shuffle/sort steps account for about 80% of the running time.

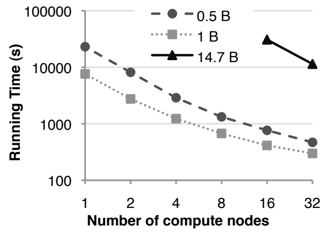


Step	Base		Improved		Speedup
	Sec	Rel %	Sec	Rel %	X
Sampling	1555	13.4%	859	22.5%	1.8
Splitting	62	0.5%	62	1.6%	1.0
Load particles	229	2.0%	232	6.1%	1.0
Load box idx	3	0.0%	12	0.3%	0.3
Box search	3422	29.5%	122	3.2%	28.0
Shuffle / sort	4363	37.6%	1000	26.2%	4.4
FOF data gen.	762	6.6%	320	8.4%	2.4
FOF exec	576	5.0%	584	15.3%	1.0
Merging	151	1.3%	137	3.6%	1.1
Relabeling	486	4.2%	482	12.7%	1.0
Total	11609	100.0%	3810	100.0%	3.0

Figure 7: Breakdown of the DiscFinder elapsed time needed to find groups in a snapshot of the DMKraken 14.7 billion particle dataset on 32 worker nodes.

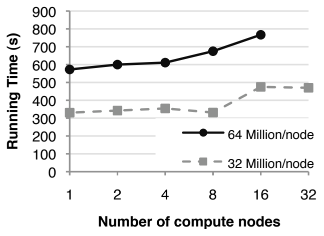
Performance Improvements. We implemented a set of optimizations to improve the overall performance of the pipeline. The breakdown of the running time for the improved version is shown in columns 4 and 5 of Figure 7. Column 4 has the absolute time for each step in seconds and Column 5 contains the relative time with respect to the total running time of the improved version. Column 6 shows the speedup for that step relative to the baseline. The overall speedup is 3X. Although the framework presents a easy-to-use programming abstraction, they may lead to inefficient implementation. Producing performant implementations becomes challenging and improving the performance requires detailed knowledge of the framework. Below are the anecdotes of our debugging experience.

Fixing the Sampling/Splitting Phase. In the initial implementation, all the tasks deterministically sampled the dataset in parallel by choosing the same set of sample points (starting with a preset seed



Nodes	0.5 B	1.1 B	14.7 B
1	23107	7625	n/a
2	8149	2754	n/a
4	2890	1234	n/a
8	1326	675	n/a
16	767	415	30816
32	470	299	11429

Figure 5: Strong scaling.



Nodes	60 M/n	32 M/n
1	573	331
2	600	342
4	611	354
8	675	331
16	767	475
32	n/a	470

Figure 6: Weak scaling.

for the random number generator). However, all the tasks were reading a lot more data than needed. This was caused by the implementation, in the framework libraries, for fetching data from the distributed file system. The improved solution consists of splitting the data range over the parallel tasks and having each task sample its subset of the data. Each task still reads too much data, but the total data read is reduced by a factor equal to the number of tasks. We are working on further reducing the sampling time by modifying the framework libraries so the amount of data read from the file system is reduced further.

Speeding up Box Lookups. The Box Lookup step determines to which partition a particle belongs. The performance of this step was affected by the initial implementation choice. Since the number of partitions is relatively small, we originally used a simple linear search mechanism. This box lookup is performed for every particle and on aggregate, the lookup time becomes significant. Replacing the lookup mechanism with a routine that uses a $O(\log n)$ algorithm, where n is the number of partitions, provided major benefits.

Adjusting Number of Reducers. In this set of experiments the particles are split into 1024 spatial partitions. At first, it was natural to set the total number of reduce tasks equal to the total number of partitions, such that a reduce task would process a partition in the same way that in computational sciences applications the number of partitions matches the number of processing units. During the execution of the original DiscFinder implementation, we noticed in our cluster monitoring tool that the network bandwidth consumption had a cyclic behavior with periods of high utilization followed by idle periods. This effect was caused by multiple waves of reduce tasks fetching data produced by the map tasks. By setting the number of reducers for the job to $(\text{numberOfNodes} - 1) * \text{reducersPerNode} = 124$, all the reduce tasks are able to fetch and shuffle the data in a single wave, even in the presence of a single node failure. In this case, each reduce task processes multiple partitions. This lesson can be applied to other similar Hadoop applications.

Overall the running time significantly decreased after these modifications. There is still room for improvement. We are working on further speeding up the running time, both at the application level in terms of the algorithmic approach and at the framework level in terms of more efficient implementations.

5. CONCLUSION

With the increasing in size of current and future astrophysics datasets, analysis tools need to scale up to operate on these massive datasets. DiscFinder is a data-intensive distributed cluster finder that can operate on tens of billions of objects, which enables the analysis of the currently largest sky surveys and cosmology simulation datasets. We evaluated its efficiency and presented approaches for improving its performance under Hadoop. We will continue developing new tools to support data-intensive applications in science.

6. REFERENCES

[1] Hadoop. `http://hadoop.apache.org`.

[2] K. Abazajian et al. The Seventh Data Release of the Sloan Digital Sky Survey. *ApJS*, 182, June 2009.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[4] T. Di Matteo et al. Direct cosmological simulations of the growth of black holes and galaxies. *ApJ*, (676), 2008.

[5] B. Fu, J. López, E. Fink, K. Ren, and G. Gibson. DiscFinder: A data-intensive scalable cluster finder for astrophisics. Technical Report CMU-PDL-2010-104, Parallel Data Laboratory, Carnegie Mellon University, 2010.

[6] B. Galler and M. Fischer. An improved equivalence algorithm. *CACM*, 7(5):301–303, 1964.

[7] J. Gardner et al. A framework for analyzing massive astrophyisical datasets on a distributed grid. In *ADASS XVI*, 2006.

[8] S. Gill et al. The evolution of substructure - I. A new identification method. *MNRAS*, 351:399–409, 2004.

[9] K. Heitmann et al. The Coyote Universe I: Precision determination of the nonlinear matter power spectrum, 2008.

[10] J. Huchra and M. Geller. Groups of galaxies. I - Nearby groups. *ApJ*, 257:423–437, 1982.

[11] Y. Kwon et al. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. Technical Report UW-CSE-09-06-01, University of Washington, June 2009.

[12] Y. Liu et al. Design and evaluation of a parallel HOP clustering algorithm for cosmological simulation. *Proc. Parallel and Distributed Processing Int. Symp.*, 2003.

[13] MPI Forum. MPI: A Message Passing Interface. In *Supercomputing '93*. ACM/IEEE, 1993.

[14] D. Pfitzner and J. Salmon. Parallel halo finding in N-body cosmology simulations. In *KDD-96*, 1996.

[15] V. Springel. The cosmological simulation code gadget-2. *MNRAS*, 364(4):1105–1134, 2005.

[16] U. Washington N-Body Shop. FOF: Find groups in N-body simulations using the friends-of-friends method.

[17] M. White. The mass function. *ApJ*, 143:241, 2002.