Proceedings of the Twenty-Third Scientific and Statistical Database Management Conference (SSDBM 2011), 20-22 July 2011.

Recipes for Baking Black Forest Databases

Building and Querying Black Hole Merger Trees from Cosmological Simulations

Julio López, Colin Degraf, Tiziana DiMatteo, Bin Fu, Eugene Fink, and Garth Gibson

Computer Science and Physics departments, Carnegie Mellon University

Abstract. Large-scale N-body simulations play an important role in advancing our understanding of the formation and evolution of large structures in the universe. These computations require a large number of particles, in the order of 10-100 of billions, to realistically model phenomena such as the formation of galaxies. Among these particles, black holes play a dominant role on the formation of these structure. The properties of the black holes need to be assembled in merger tree histories to model the process where two or more black holes merge to form a larger one. In the past, these analyses have been carried out with custom approaches that no longer scale to the size of black hole datasets produced by current cosmological simulations. We present algorithms and strategies to store, in relational databases (RDBMS), a forest of black hole merger trees. We implemented this approach and present results with datasets containing 0.5 billion time series records belonging to over 2 million black holes. We demonstrate that this is a feasible approach to support interactive analysis and enables flexible exploration of black hole forest datasets.

1 Introduction

The analysis of simulation-produced black hole datasets is vital to advance our understanding of the effect that black holes have in the formation and evolution of large-scale structures in the universe. Increasingly larger and more detailed cosmological simulations are being used to gain insight on the evolution of massive black holes (Sec. 2). The simulations store the data in a format that is not readily searchable or easy to analyze. Purpose-specific custom tools have often been preferred over standard relational database management systems (RDBMS) for the analysis of datasets in computational sciences (Sec. 3). The assumption has been that the overhead incurred by the database will be prohibitive. Previous studies of black holes have used custom tools. However, this approach is inflexible as these tools often need to be re-developed for carrying out

Acknowledgments: This research was sponsored in part by the National Science Foundation (NSF), under award CCF-1019104; the Gordon and Betty Moore Foundation, in the eScience project; and the support of the companies of the PDL Consortium. The computations were supported by an allocation of advanced computing resources supported by the NSF and the TeraGrid Advanced Support Program. Computations were performed at the Carnegie Mellon Cloud Cluster (wiki.pdl.cmu.edu/opencloudwiki) and the National Institute for Computational Sciences.

new studies and answering new questions. As part of our goal of reducing the time to science, we developed an approach that leverages RDBMS to analyze black hole datasets (Sec. 4). This approach enables fast, easy and flexible data analysis. A major benefit of the database approach is that now the astrophysicists are able to interactively ask ad-hoc questions about the data and test hypotheses by writing relatively simple queries and processing scripts. We present: (1) A set of algorithms and approaches for processing, building and querying black hole merger tree datasets. (2) A compact database representation of the merger trees. (3) An evaluation of the feasibility and relative performance of the presented approaches. Our evaluation (Sec. 5) shows that it is feasible to support the analysis of current black hole datasets using a database approach. An extended version of the results presented here is also available [13].

2 Motivation: Black Holes and the Structures in the Universe

Black holes play an important role in the process by which structures, such as galaxies, are organized in the universe. To understand these phenomena, large-scale cosmological numerical simulations are used. They cover a vast dynamic range of spatial and time scales with an extremely large number of particles, in excess of 10^{10} in principle.

Black Hole Datasets. The simulations produce three types of datasets: snapshots, group membership and black holes. Snapshots contain complete information for all the particles in the simulation at a given time step. In recent simulations, snapshots require close to 100 TB of storage. The group files contain the membership of particles to groups, such as dark matter halos. The black hole files contain the black hole data with high temporal resolution. They contain two main types of records. (1) Black hole property records contain the id, simulation time, mass, and other properties. (2) Merger events records indicate when a pair of black holes merge with one another and contain the ids and masses of the two black holes, as well as the time when the event occurred. A *black hole merger tree* comprises the set of merger event records along with the detailed property records for the black holes involved in the mergers.

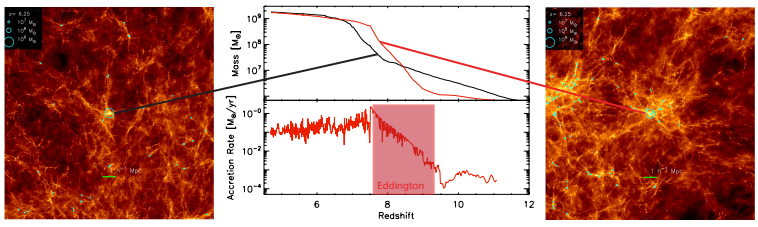


Fig. 1. Sample black holes. This figure shows the gas distribution around two large black holes and their respective light curves and accretion rate history for the most massive one.

Analysis of Black Hole Datasets. Recent observations imply that black holes with billion solar masses are already assembled when the universe is only 800 million years old.

An objective of the analyses of simulation-generated black hole datasets is to explain the formation of these objects. There are two types of analyses we want to perform on black hole datasets. The first type requires queries based on a specific redshift (i.e., simulation time), often selecting a subset according to their mass and growth rate. These analyses aim to characterize the properties of black hole properties that exist at a specific time, including the number and density of black holes as a function of mass [6] or luminosity [4], how they cluster and the correlation between black holes and the galaxies in which they are found [3,6,5]. The second type of analyses requires processing the detailed growth history of individual black holes. An example is shown in Fig. 1. These histories help us understand how black holes grow, the relative importance of black hole mergers vs. gas accretion.

3 Background and Related Work

Database techniques have been adopted to manage and analyze datasets in a variety of science fields such as medical imaging [2], bioinformatics [15] and seismology [16]. In astronomy, RDBMS have been used to manage the catalogs of digital telescope sky surveys such as the Sloan Sky Digital Survey (SSDS) [1,9]. Database techniques have been used in observational astronomy to perform anomaly detection [10] among others, and data-intensive approaches have been used for spatial clustering [7,11]. RDBMS have not been as widely used for the analysis of cosmological simulations, in part due to the challenge posed by the massive multi-terabyte datasets generated by these simulations. The German Astrophysical Virtual Observatory (GAVO) has led in this aspect by storing the Millenium Run dataset in an RDBMS and enabling queries to the database through a web interface [12]. GAVO researchers proposed a database representation for querying the merger trees of galactic halos. We are using RDBMS to support interactive analysis of cosmological simulation datasets. We present techniques for building and querying the merger trees of black holes, along with a compact database representation for these trees.

4 Building and Querying Black Forest Databases

Database Design. To support the queries needed for the analysis of BH datasets, we transform the the simulation output into RDBMS tables. The database comprises two main tables as shown in Fig. 2: *BlackHoles* (BH), *MergerEvents* (ME). Querying this database consists of two steps: (1) building the merger tree from the ME table to obtain the ids of the black holes in the tree; (2) querying the BH table to retrieve the associated history for the black holes. The input for a query is the id of a black hole of interest (`qbhid`). The desired output for step 1 is the ids of all the black holes in the same merger tree as `qbhid`. Notice that the ME records do not have explicit links to other ME records that belong to the same merger tree. The approaches for building and querying the merger trees are presented below.

Approach 1: Recursive DB Queries. Given a `qbhid`, this approach finds the root of the tree by repeatedly querying the ME table. Once the root is found, it recursively queries

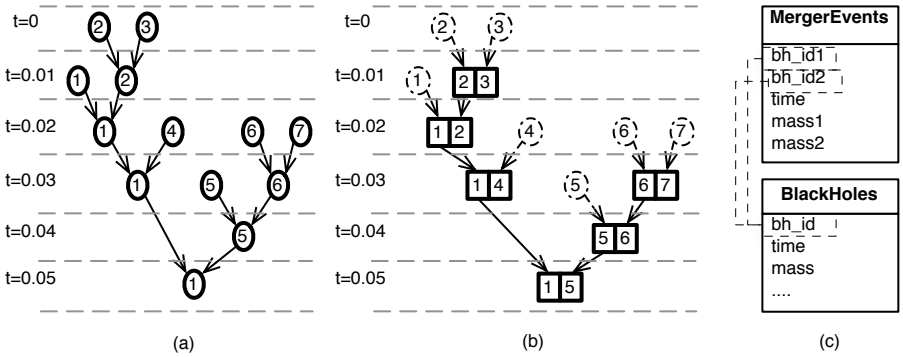


Fig. 2. (a) Black hole merger tree. Leaf nodes (at the top) correspond to black holes. Interior nodes correspond to black holes that merge. (b) DB representation: only the interior nodes of the tree, i.e., merger events, are stored, the dashed circles, corresponding to the leaf nodes, are not explicitly stored. (c) Basic schema for main tables in the black holes database: mergerevents (ME) and blackholes (BH).

the ME table for each of the root's children (`left`, `right`) as shown in the `BuildTree` procedure. This simple approach works well when only a small number of merger trees are being queried and the resulting trees have few records.

Procedure BuildTree(bhroot, ctime): *Recursively build a merger tree rooted at bhroot*

```
   // Find all the records that have the bh1 field = bhroot
 1 type TreeNode {id, time, left, right }
 2 TreeNode node = NULL, pnode = NULL
 3 qresult = SELECT bh2, time FROM ME WHERE bh1 = bhroot AND time ≤ ctime
   ORDER BY time DESC
 4 for (bh2, time) in qresult do
 5     node = new TreeNode(id, time)
 6     node.right = BuildTree(bh2, time)
 7     if pnode is not null then
 8         pnode.left = node // set left child for previous node in the result
 9     pnode = node
10 return node // node is the latest event (tree root), it may be null
```

Approach 2: In-Memory Queries. This approach consists in using a single database query for loading all the records from the ME table into a set in memory (`MESet`) and then looking up in `MESet` the events that belong to a tree. The algorithm is the following. Given a query `qbhid`, add it to a queue `pq` of pending black holes. For each element `bh` in the queue, fetch from `MESet` the records that match `bh` (i.e., `r.bh1 = bh`). For each matching record `r`, add the corresponding `r.bh2` to the `pq` queue. Repeat this process

until every element of pq has been processed (i.e., the end of the queue is reached). At the end of the procedure, pq contains the ids belonging to the corresponding tree.

Approach 3: In-Memory Forest Queries. This approach builds on the previous one. The basic idea is to build all the merger trees in the dataset with a single scan of the ME table, instead of building a single tree as in the previous approach. This approach incurs extra work to build all the trees. However, this cost is amortized when a large number of queries need to be processed. This approach is based on the Union-Find algorithm [8] and adjusted to handle the peculiarities of the merger events representation. The process is described in the procedure `BuildForestInMemory`.

Procedure BuildForestInMemory(db)

```
   input  : DB with the ME table
   output : A forest containing all the merge trees in ME
 1 cursor = SELECT bh1, bh2, time FROM ME // Scan over all ME records
 2 for (bh1, bh2, time) in cursor do
 3     node = new TreeNode(bh1, time, bh2)
 4     bh2Map.put(bh2, node) // Map from bh2 to this node
 5     bh1Map.addToList(bh1, node) // Map from bh1 to a node list
 6 for node in bh2Map do
 7     node.right = bh1Map.get(node.bh2) // Create link for right-side child, it may be null
 8 forest = emptySet()
 9 for lst in bh1Map do
10     sortbytime(lst)
11     createLinkOnBh1(lst) // Create links from lst[n-1].left to lst[n]
12     findRootAndAddToForest(lst, forest)
13 return (forest, bh1Map, bh2Map)
```

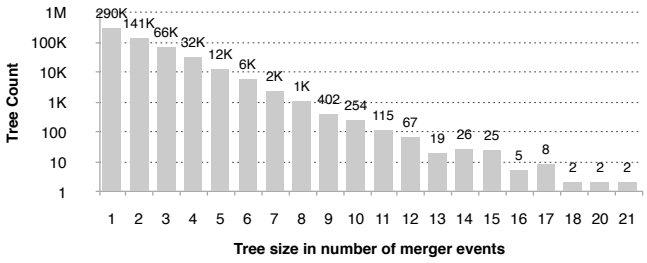
Approach 4: ForestDB. The ForestDB approach builds on the techniques used in the In-Memory Forest approach. The basic idea is to build the black hole forest in the same way as in the in-memory case. Then tag each tree with an identifier (`tid`). The forest can be written back into a table in the database that we will call merger events forest (MF). This is done as a one-time pre-processing step. The schema for this table is the same as the ME's schema (see Fig. 2), with the addition of the `tid` field. Two conceptual steps are performed at query time to extract a merger tree for a given `qbhid`. First, search the MF table for a record matching `qbhid`. The `tid` field can be obtained from the record found in this step. Second, retrieve from the MF table all the records that have the same `tid`. These two steps can be combined in a single SQL query. Moreover, the detailed history for the black holes in the tree can be retrieved from the BH table using a single query that uses `tid` as the selection criteria and joins the MF and BH tables. Indices on the `bh1`, `bh2` and `tid` fields are required to speed up these queries. Alternatively, the indices on `bh1` and `bh2` can be replaced by an additional auxiliary indexed table to map from `bhid` to `tid`.

The MF table only stores the membership of the merger event records to a particular tree. Notice that the MF table does not explicitly store the tree structure, i.e., the parent-child relationships. Also, the MF table only stores the internal nodes of the merger tree. The leaves are not explicitly stored. Instead the relevant data (such as the leaf's `bhid`) is stored in the parent node. This makes for a more compact representation as it requires fewer records in the MF table.

5 Evaluation

We implemented the approaches described above using Python and SQLite. Our evaluation aims to characterize the relative performance of these approaches and determine the feasibility of using RDBMSs in the analysis of black holes datasets. For this purpose, we ran a set of experiments using a dataset produced by the largest published cosmology simulation to date.

Workload. The dataset was produced by a cosmological simulation using the GADGET-3 [14] parallel program. The simulation contained 66 billion particles. At the end of the simulation, there are 2.4 million black holes. The size of the resulting black holes dataset is 84 GB. The black hole history table contains 420 million records corresponding to 3.4 million unique black holes and 1 million merge events. Figure 3 shows the distribution of tree sizes in number of merger events in the ME table. The storage requirements for the tables and associated indexes is shown in the table in Figure 3.



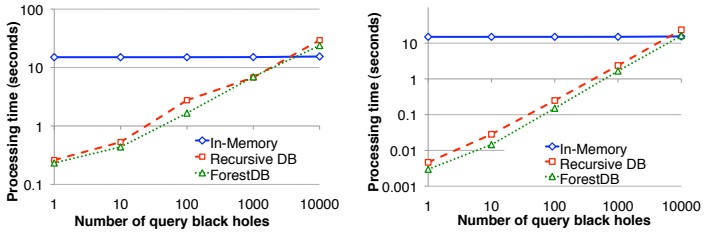
BH dataset table sizes			
Table	Rec. count	Tab. size	Idx. size
BH	420 M	50 GB	22 GB
ME	1 M	49 MB	26 MB
MF	1 M	53 MB	27 MB

Fig. 3. Left: Distribution of tree sizes in the black holes dataset. The X axis is the size of a merger tree measured as the number of events in a tree. The Y axis is the number of trees of that size in $\log_{10}$ scale. Right: Sizes of tables and indexes in the BH database.

Performance. To characterize the performance of the developed approaches, we conducted a series of micro benchmark experiments that correspond to the steps involved in answering queries for the detailed time history of merger trees. The experiments were run on a server host with 2 GHz CPUs, 24 GB of memory and a SATA disk.

Building Merger Trees. The first set of micro benchmark experiments corresponds to the steps needed to build the merger trees for a set of query black holes (qbhs). We compared three of the approaches explained in Sec. 4: (a) Recursive DB – RDB, (b) In-memory – IM, and (c) Forest DB – FDB. The In-memory Forest approach was only used

to build the tables for FDB. For these experiments we selected black holes (*qbhs*) that belonged to merger trees in the ME table. We timed the process of satisfying a request to build one or more merger trees specified by the requested qbhs. The processing time includes the time required to issue and execute the database query, retrieve and post-process the result to build the trees.



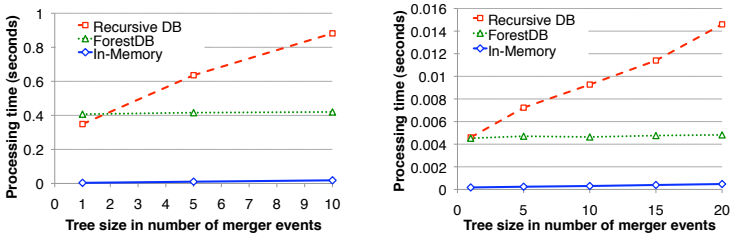
(a) Cold OS cache (b) Warm OS cache

Fig. 4. Running time to obtain the merger trees for the different approaches. These results correspond to a tree of size 5. The X axis is the number of trees being queried at once in a batch. The Y axis is the elapsed time in seconds (log scale) to retrieve the corresponding records from the ME table. The cases with cold (a) and warm (b) OS caches are shown.

In the first experiment, we kept the tree size fixed at 5 and varied the number of black holes for which a tree is requested (number of qbhs). The results for the different approaches are shown in Fig. 4. The X axis is the qbh count varying from 1 to 10K. The Y axis shows the processing time (seconds) in log scale. For qbh counts less than 1K, both the RDB and FDB approaches are faster than the In-Memory approach. The RDB approach is not as expensive as we originally thought for small queries, either in the number of qbhs or the requested tree size. It was surprising to find out that for the cold OS cache setup (Fig. 4a), the processing time for RDB and FDB does not differ significantly. For the warm OS cache, there is a (constant in log scale) difference between RDB and FDB. The IM approach pays upfront a relatively large cost of 15 seconds to load the entire ME table, then the processing cost per requested qbh is negligible, and thus can be amortized for a large number of qbhs.

Figure 5 shows the effect of the merger tree size on the request processing time. In this experiment the requests were grouped by tree sizes (X axis = 1, 5, 10, 15, 20). This experiment was performed with a warm OS cache and cold database cache. The initial load time for the IM approach is not included in the processing time shown in the graphs, only the time to build the tree in memory. The running time for the RDB approach increases as the trees get larger. This is due to the larger number of queries to the ME table needed to process each tree in the recursive approach. The FDB approach requires a single query to the ME table per requested tree.

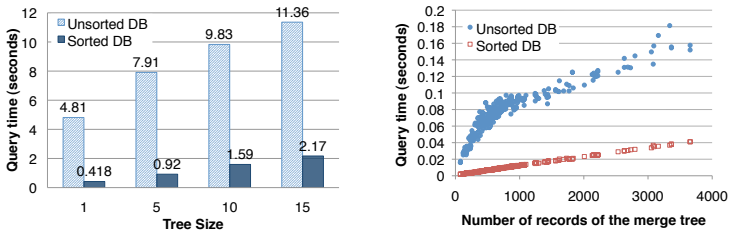
Retrieving the Time History for Merger Trees. In the second set of experiments, we retrieved the detailed time history for a set of trees retrieved in the previous step. This entails retrieving from the BH table all the records for the corresponding BH in a given



(a) Batch size=250 qbhs (b) Batch size=2 qbhs

Fig. 5. Processing time for building the merger trees using various approaches. This experiment was performed with a warm OS cache and a cold DB cache. The X axis is the size of the resulting tree; (a) and (b) show the time to process 250 qbhs and 2 qbhs per request respectively. The Y axis is the elapsed time to build the number of trees of each size.

merger tree. For each tree size (1, 5, 10, 15), we retrieved the BH histories for 100 trees of that size. Figure 6a shows the elapsed time in seconds to retrieve the detail records from the BH table. The times are shown for an unsorted indexed BH table and a BH table sorted by the black hole id. As expected for this query pattern, sorting by the BH id is beneficial. Figure 6b shows the elapsed time according to the number of records that were retrieved from the BH table. Each data point corresponds to a merger tree that resulted in retrieving the number of BH records shown in the X axis. The Y axis is the elapsed time in seconds for the unsorted and sorted BH tables.



(a) (b)

Fig. 6. Time to retrieve the detail BH history from the BH table for merger trees of various sizes. The running times for queries to sorted and unsorted BH tables are shown. Figure (a) shows the elapsed time grouped by tree size. Figure (b) shows the same data grouped by the number of BH records comprising the merger trees.

6 Conclusion

Rapid, flexible analysis of black hole datasets is key to enable advances in astrophysics. We presented a set of algorithms for processing these data using a database approach.

The database approach is not only flexible, but also exhibits good performance to support interactive analysis.

References

1. ABAZAJIAN, ET AL. 7-th Data Release of the Sloan Digital Sky Survey. *ApJS 182* (2009).
2. COHEN, S., AND GUZMAN, D. E. SQL.CT: Providing data management for visual exploration of ct datasets. In *SSDBM: Scientific and Statistical Database Management* (2006).
3. COLBERG, J. M., AND DI MATTEO, T. Supermassive black holes and their environments. *Monthly Notices of the Royal Astronomical Society (NMRAS) 387* (July 2008), 1163–1178.
4. DEGRAF, C., ET AL. Faint-end quasar luminosity functions from cosmological hydrodynamic simulations. *Monthly Notices of the Royal Astronomical Society (NMRAS) 402* (2010).
5. DEGRAF, C., ET AL. Quasar Clustering in Cosmological Hydrodynamic Simulations: Evidence for mergers. *ArXiv e-prints* (2010).
6. DI MATTEO, T., ET AL. Direct Cosmological Simulations of the Growth of Black Holes and Galaxies. *Astrophysical Journal (ApJ) 676* (2008), 33–53.
7. FU, REN, LÓPEZ, FINK, AND GIBSON. Discfinder: A data-intensive scalable cluster finder for astrophysics. In *High Performance Distributed Computing (HPDC)* (2010).
8. GALLER, AND FISCHER. An improved equivalence algorithm. *Comm. ACM 7*, 5 (1964).
9. IVANOVA, ET AL. MonetDB/SQL meets skyserver: the challenges of a scientific database. In *Scientific and Statistical Database Management (SSDBM)* (2007), p. 13.
10. KAUSTAV DAS, J. S., AND NEILL, D. Anomaly pattern detection in categorical datasets. In *Knowledge Discovery and Data Mining (KDD)* (2008).
11. KWON, Y., ET AL. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. In *Scientific and Statistical Database Management (SSDBM)* (2010).
12. LEMSON, AND SPRINGEL. Cosmological simulations in a relational database: Modelling and storing merger trees. In *Astronomical Data Analysis Software and Systems* (2006).
13. LOPEZ, ET AL. Recipes for baking black forest databases. Tech. Rep. CMU-PDL-11-104, Carnegie Mellon, PDL, 2011.
14. SPRINGEL, V. The cosmological simulation code GADGET-2. *Monthly Notices of the Royal Astronomical Society (NMRAS) 364* (Dec. 2005), 1105–1134.
15. XU, ET AL. Covariant evolutionary event analysis for base interaction prediction using a relational database management system for RNA. In *SSDBM* (2009).
16. YANG, Y.-S., ET AL. Isee: Internet-based simulation for earthquake engineering - Part I: Database approach. *Earthquake Engineering & Structural Dynamics 36* (2007), 2291–2306.