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 Garth Gibson

CMU-PDL-11-104

April 2011

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

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 Parallel Data Laboratory Consortium (PDL) at Carnegie Mellon University (CMU). The computations were supported by an allocation of advanced computing resources supported by the NSF and the TeraGrid Advanced Support Program. The computations were performed on on Kraken at the National Institute for Computational Sciences (NICS) http://www.nics.tennessee.edu and the CMU Open Cloud Cluster http://wiki.pdl.cmu.edu/opencloudwiki

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. Computational cosmologists are interested in the analysis of black hole properties throughout the simulation with high temporal resolution. 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 this analysis has been carried out with custom approaches that no longer scales to the size of black hole datasets produced by current cosmological simulations. We present a set of algorithms and a strategy to represent and store a forest of merger trees for black holes in relational databases (RDBMS). 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 developed and carried out to increase the statistical significance of the generated particle and black hole datasets. These higher resolution datasets are key to gain insight on the evolution of massive black holes.

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. 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. The tools often need to be re-developed for carrying out new studies and answering new questions. We recently faced this challenge when the existing tools could not handle the data sizes produced by our recent simulations. As part of our goal of reducing the time to science, we decided to leverage RDBMS implementations to perform the analysis of black hole datasets. 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:

- A set of algorithms and approaches for processing, building and querying black hole merger tree datasets.
- A compact database representation of the merger trees.
- An evaluation of the feasibility and relative performance of the presented approaches.

Our evaluation suggests that it is feasible to support the analysis of current black hole datasets using a database approach. The rest of this paper is structured as follows. Section 2 describes our motivating science application, the analysis of black hole datasets. Section 3 contains background information and related work. Section 4 presents various approaches for processing black hole datasets. The evaluation follows in Section 5. Sections 6 and 7 discuss the future work and conclusions.

2 Black Holes and the Structures in the Universe

Black holes play an important role in the process by which large-scale structures, such as galaxies, groups and clusters of galaxies, are organized in the universe. Structure formation and evolution in cosmology encompasses the description of the rich hierarchy of structures in the universe, from individual galaxies and groups to clusters of galaxies up to the largest scale filaments along which smaller structures align. To study this process, cosmological numerical simulations that cover a vast dynamic range of spatial and time scales are being developed. These need to include the effect of gravitational fields generated by superclusters of galaxies on the formation of galaxies, which in turn harbor gas that cools and makes stars and is being funneled into supermassive black holes the size of the solar system.

There are two conflicting requirements that make the study of hierarchical structure formation extremely challenging. To have a statistically significant representation of all structure in the universe, the studied volume needs to be large. However, the particle mass needs to be relatively small to adequately resolve the appropriate physics and the scale of the structures that emerge. This implies a need for an extremely large number of particles in the simulation, requiring in principle a dynamic range of 10^{10} or more.

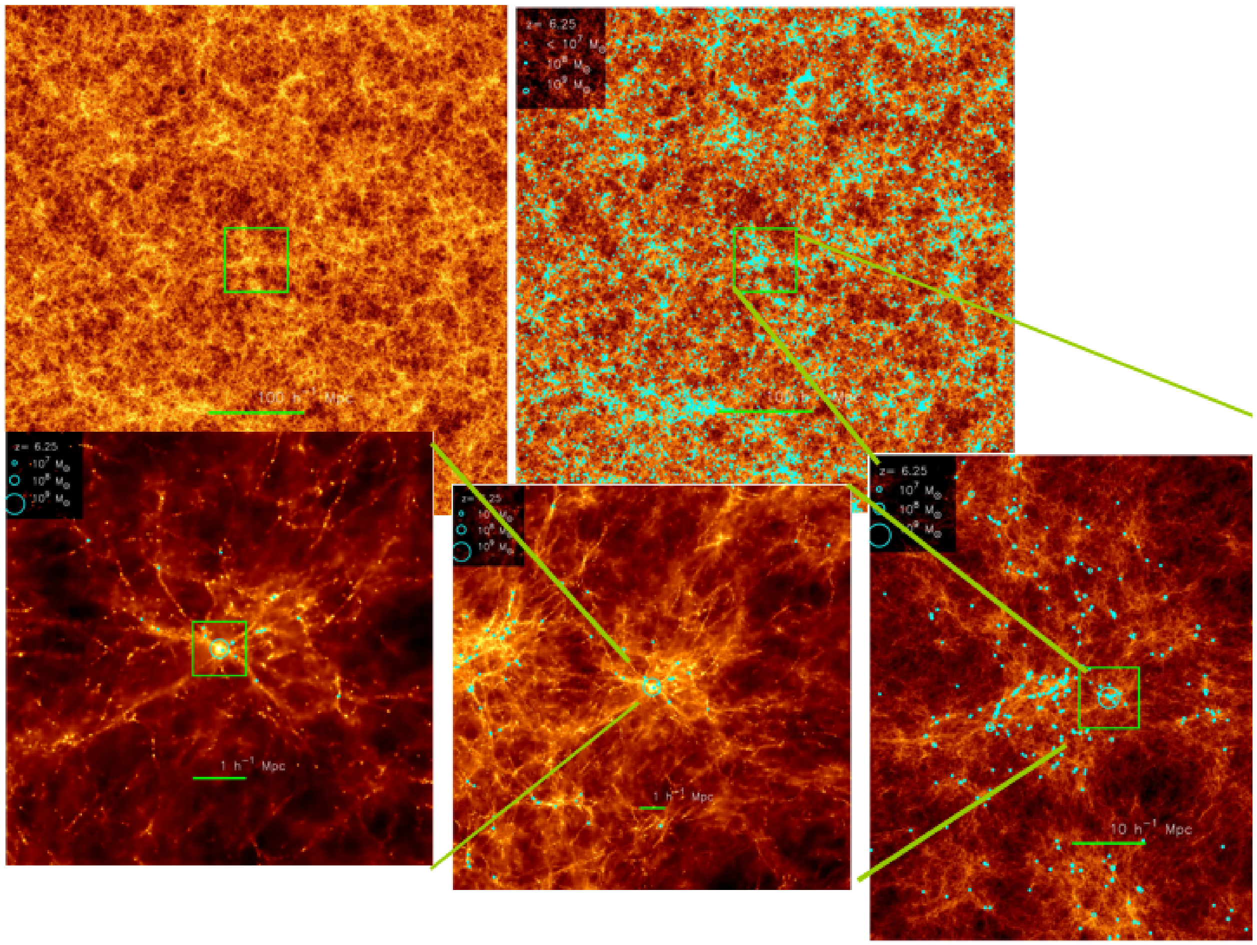


Figure 1: Visualizations of a large-scale dark matter simulation carried out by the researchers of the McWilliams Center for Cosmology. The top two pictures show slices of the simulation volume. The black holes are shown as bright dots (light color). Successive zoom factors (10X) of the highlighted box are shown in the bottom 3 frames.

Scientists at the CMU McWilliams Center for Cosmology and collaborating institutions use the parallel program GADGET-3 [28] to carry out large scale cosmology simulations on supercomputers with 100K CPU cores. A visualization of the result of these computations is shown in Fig. 1. The simulations evolve an initial realization of a Lambda Cold Dark Matter (ΛCDM) cosmology over cosmological timescales, incorporating dark matter, gas, stars and black holes. The gravitational forces are calculated with a hybrid approach, named TreePM, that combines a hierarchical tree algorithm for short range forces with a particle-mesh algorithm for long-range forces. The gas is modeled using Smoothed Particle Hydrodynamics (SPH) which uses the Lagrangian method of discretizing mass into a set of particles with adaptive smoothing lengths [30], naturally providing a varying resolution from the densest regions at the center of massive galaxy halos to the diffuse voids between halos. Sub-resolution models are used to model star formation and supernova feedback (using the multi-phase model of [31]) as well as black hole formation, accretion, and feedback [29, 11].

Analysis of Black Hole Datasets. With the current simulation capabilities, we are now in a position to make predictions about the mass distribution in the inner regions of galaxies. In particular, recent observations imply that black holes with billion solar masses are already assembled and seen as the first quasars and large galaxies when the universe is only 800 million years old. As these objects are likely to occur in extremely rare high density peaks in the early universe, large computational volumes are needed to study them. An aim of high-resolution, large-volume simulations and associated analysis of the produced datasets is to explain the formation of these objects, which is a major outstanding problem in structure development.

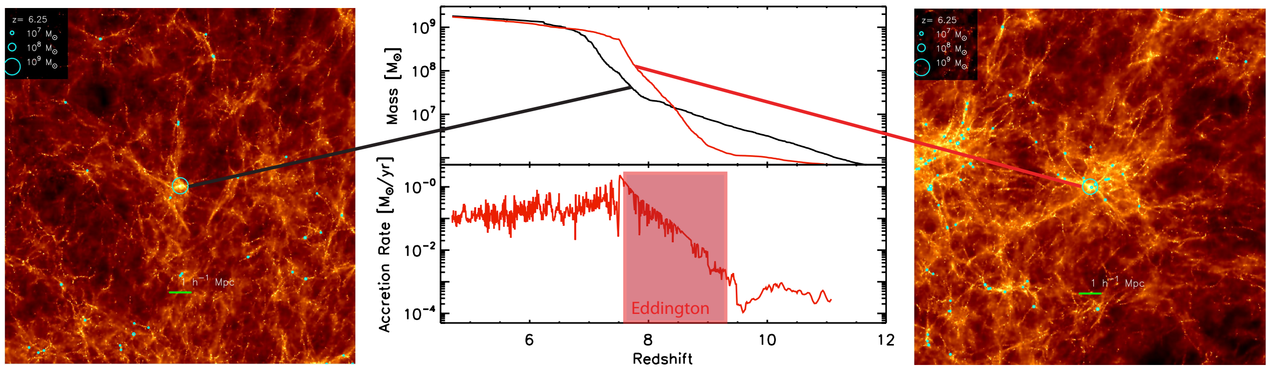


Figure 2: Sample black holes. This figure shows the gas distribution around two of the largest black holes in a snapshot from a recent simulation. The respective light curves for these black holes are shown in the plot, as well as the accretion rate history for the most massive one.

There are two general types of questions which we typically want to address using the black hole datasets. The first one is to investigate the black hole populations which exist at a specific time. These queries on the overall population of black holes have been used to study a wide variety of black hole properties, such as the number and density of black holes as a function of mass [10] or luminosity [8], the clustering properties of different populations of black holes [9], and the correlation between black holes and the galaxies in which they are found [6, 10]. These questions require queries based on a specific redshift (i.e., simulation time), often selecting a subset of the black holes at that time based on their mass and accretion rate. The second type of questions requires looking at the detailed growth history of individual black holes. An example is shown in Fig. 2. These histories can help us understand how black holes grow, the relative importance of black hole mergers vs. gas accretion, how the black hole luminosity varies with time, and how they depend on their surrounding environment [6, 10]. At a high-level these analyses involve deriving, examining and correlating aggregate statistics of the black hole datasets obtained from the simulations. In particular, it requires obtaining the time history for a black hole including the information about which black holes merged.

Black Hole Datasets. The simulations produce three types of datasets: snapshots, group membership and black holes. The snapshots contain complete information for all the particles in the simulation at a given time step. Snapshots have a high cost in terms of both time and storage space. For example, the snapshots of the latest simulations are each 3 TB in size. Only a few snapshots are stored per simulation (e.g., 30). The output frequency varies throughout the simulation, with snapshots being written more frequently at later times when the simulation exhibits a highly non-linear behavior. The group files contain the statistics of cluster structures, such as dark matter halos, as well as the group membership information for particles in the snapshots.

In addition, the black hole data is written to a separate set of text files at a much higher time-resolution to preserve the black hole information. This is possible because the black holes only make up a relatively small fraction of the total number of particles in the simulation. Each of the compute hosts participating in the simulation produces a file containing the data associated with the black holes residing in that host. The files contain one of the following three types of records per line: black hole properties, near mergers, and merger events. (1) The black hole properties are stored when they are re-calculated for a new time step. The stored properties include the id, simulation time, mass, accretion rate, position, velocity relative to the surrounding gas, local gas density, local sound speed, and local gas velocity. Records of this type make the majority of the black hole output. (2) Near merger records are produced when a pair of black holes are close enough to initiate a merger check, but are moving too rapidly past one another. The output record contains the simulation time, each black hole id, the velocity of the black holes relative to one another, and the local sound speed of the gas. (3) Merger event records are stored when a pair of black holes merge with one another. The output record contains the ids and masses of the two black holes and the time at which the merger 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.

3 Background and Related Work

The scientific computing community has developed several file formats, such as FITS [36], NetCDF [25] and HDF5 [12], for storing datasets produced by numerical simulation, atmospheric and astronomy observations. These formats support efficient storage of the dense array-oriented data. However, these formats have limited mechanisms for indexing objects based on their values and fast retrieval of matches to specific queries. Specialized applications, e.g., in seismology [27, 35] and geographical information systems (GIS), have used indexing structures such as B-trees [3], R-trees [15], octrees [26, 34] and *kd*-trees [20]. The use of RDBMSs for array oriented data, in systems such as rasdaman [2], is an emergent area of active research.

Database techniques have been adopted to manage and analyze datasets in a variety of science fields such as neuroscience [22], medical imaging [5], bioinformatics [37] and seismology [38]. In astronomy, RDBMSs have been used to manage the catalogs of digital telescope sky surveys [17, 4]. For example, the Sloan Sky Digital Survey (SSDS) has collected more than 300 million celestial objects to date [1]. Database techniques have been used in observational astronomy datasets to perform spatial clustering [32] and anomaly detection [18] among others. Various research groups have used distributed computing frameworks, such as MapReduce [7], Pig [24] and Dryad [16] for clustering and analysis of massive astrophysics simulations [23, 13, 19].

RDBMSs 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 [21]. GAVO researchers proposed a database representation for querying the merger trees of galactic halos.

In our collaboration with astrophysicists, we are using RDBMSs to support the analysis of cosmological simulation datasets. We present various techniques for building and querying the merger trees of black holes. We present a modified database representation for these trees that is compact and addresses the particular requirements of the black hole datasets produced by cosmological simulations. Union-Find is an algorithm that has near-linear running time and can be used to find the connected components of a graph [14, 33]. Various approaches presented here are based on Union-Find, with adaptations to handle the specifics of the merger events representation produced by cosmological simulations. The non-RDBMS approach used to analyze previous black holes datasets is described below.

3.1 Non-DB Approach: Custom Binary Format

The analyses of black hole datasets produced by previous simulations used custom tools that employed a storage layout specifically designed for this purpose. The process comprises two steps: first generating a binary file containing all the black hole information, and then using tools to extract desired black hole data from that files. The binary file consists of a series of arrays, starting with the general properties at each timestep, i.e. the number of black holes active at any given timestep, a list of those black holes, and what the previous/next timesteps are. Then there are a series of arrays, each containing one element for every timestep of every black hole. Most of these arrays contain the basic properties of the black holes as output by the simulation (mass, accretion rate, position, etc.) but there is also an array generated with the binary file, containing the array location of the black hole's progenitor at the previous timestep (or progenitors if the previous time step contained a merger with another black hole). This array acts as a form of indexing which can be used when querying the history of a given black hole by pointing directly to the previous timestep. However, it is only helpful for this specific query, and is inefficient as it does not exploit locality of reference.

The system was quite inflexible, as the queries were hardcoded into the program, only allowing three specific requests: (a) extracting the black hole properties at a specified time, (b) extracting the complete histories of all the black holes found in the merger tree of a given black hole, and (c) extracting the most-massive progenitor history for a given black hole (i.e. the history of the most massive black hole involved in each merger event, rather than the history of every black hole). Because these queries are hardcoded into the software and rely upon the exact structure of the binary file (particularly the index pointing to the black hole(s) at the previous timestep), any other type of query required modifying the code, and would likely be very inefficient unless the binary file was re-produced with additional indexing arrays customized for that query.

The process for a user to execute queries was rather cumbersome. After producing the binary file, the user could immediately query the black holes for a specific time (query a), but the histories (queries b and c) could not be queried by black hole ID, but rather required the array index within the binary file, which could only be extracted via query (a). Thus the user must first get the list of all black holes at the end of the simulation, get the array location for the desired black hole from that list, and use that to query for the desired history. This became more involved with our new simulation, since the complete dataset could not be handled with a single binary file, instead requiring a series of binary files. Extracting the history for a single black hole thus required sequentially querying each file, with each successive file needing to be queried for all the black holes found to be part of the merger tree in the previous file, making the querying system significantly more time-consuming and error prone.

4 Building and Querying Black Forest Databases

To support the types of queries described in Section 2, we first need to transform the data produced by the simulation into a tabular, relational representation suitable for use in an RDBMS. Then, the data analysis is carried out by querying the database and processing the results using the algorithms described below.

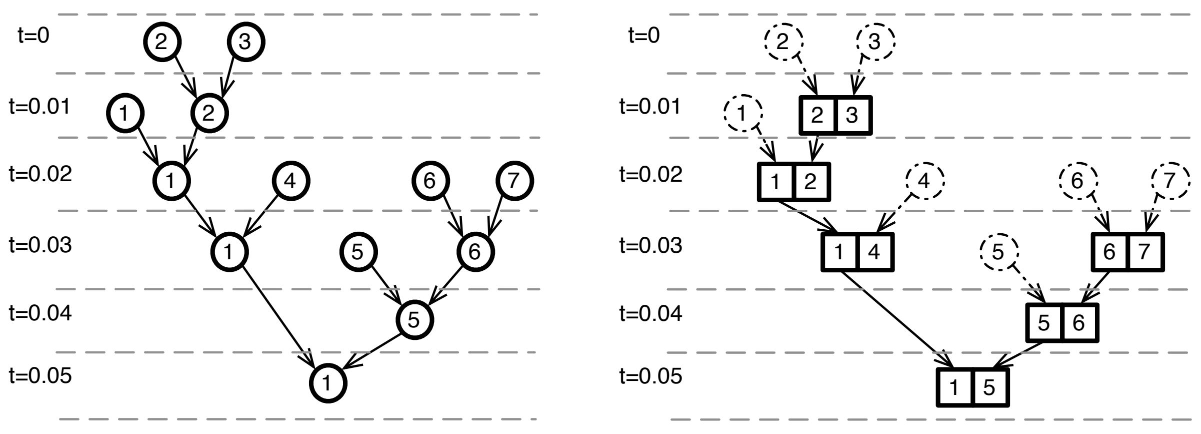


Figure 3: Merger tree representations.

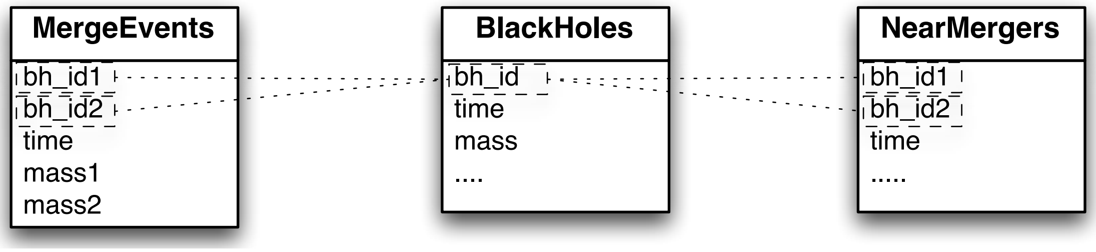


Figure 4: Basic schema for the black holes database. The MergeEvents (ME) table contains the ids and masses of the black holes that merged (`bh_id1, bh_id2, mass1, mass2`), and the `time` of the event. The BlackHoles (BH) table contains the high-resolution time history of all the black holes in the dataset. The bh_id1 and bh_id2 fields in the ME table are used as search keys to retrieve the history of the respective black holes from the BH table.

4.1 Database Design

A simple schema to represent the black holes dataset comprises three main tables as shown in Figure 4: *BlackHoles* (BH), *MergerEvents* (ME), *NearMergers* (NM). These tables correspond to the three different types of entries present in a black hole dataset produced by a simulation. Splitting the black hole output dataset into these tables is achieved through a set of pre-processing scripts that cleanup, transform and build the database. The database also contains auxiliary indices and tables to keep summary information and track particle provenance, e.g., which processor hosted the particle, which file a record came from. Storing the merger event records in a separate table enables fast construction of the merger trees. In general, the merger events account for a small size of the black holes dataset. In many cases the ME table will fit entirely in the page cache or can be loaded into the application memory for processing. Notice that the ME records do not have explicit links to other ME records that belong to the same merger tree. Different approaches for building and querying the merger trees follow.

4.2 Approach 1: Recursive DB Queries

The types of analysis described in Section 2 require retrieving the detailed time history for the black holes of interest that make up a merger tree. The procedure consists of two conceptual 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. For explanation purposes, assume that 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`. This can be easily generalized to build multiple merger trees (from a list of black holes of interest), and to retrieve a subset of the tree or the most massive progenitor path (MMPP).

The recursive DB approach works as follows. Given a `qbhid`, find the root of the tree by repeatedly querying the ME table. At each step, search for a record where `bh2` equals the current value of `bh`. Once the root is found, recursively query the ME table for each of the root's children (`left`, `right`) as shown in the `BuildTree` procedure. In the `BuildTree` procedure, `qresult` contains the left-most path for the subtree with the given root. The right child for each node in `qresult` is recursively added in the loop that iterates over `qresult`. Indices on the `bh1` and `bh2` fields are needed to speed up the queries.

This simple approach works well when only a small number of merger trees are being queried and the resulting trees have few records. However, it requires repeated queries to the database. The number of required queries is in the range $[m, 2m]$, where m is the number of merger events in the tree. In most implementations, the repeated queries will dominate the running time due to the relatively high per-query cost involved in the communication with the database engine, query parsing and execution.

```
Type TreeNode { id, time, left, right }

Procedure BuildTreeDB(qbhid):
Input: qbhid is the query black hole id
Output: the merge tree that contains qbhid

  // Find the root node of the merge tree
  bh = qbhid
  while bh is not null
    bhroot = bh
    bh = SELECT bh1 FROM ME WHERE bh2 = bh

  // Now bhroot is the root node of the merge tree
  // Then recursively build the merge tree
  return Buildtree (bhroot, inf)  // inf signals the simulation end time
```

```
Procedure BuildTree(bhroot, ctime):
// Recursively build a merger tree with bhroot as the root

  // Find all the records that have the bh1 field = bhroot
  qresult = SELECT bh2, time FROM ME
      WHERE bh1 = bhroot AND time <= ctime ORDER BY time DESC
  node = NULL
  for (bh2, time) in qresult
    node = new TreeNode(id, time)
    node.right = BuildTree(bh2, time)
    if pnode is not null
      pnode.left = node  // set left child for previous node in the result
    pnode = node

  return node  // node may be null
```

4.3 Approach 2: In-Memory Queries

This approach consists in 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 merger tree. The resulting ids are used to query the BH table in order to retrieve the detailed history for the black holes.

This approach issues a single query to the ME table, thus amortizing the initial query processing cost. On the downside, it incurs larger memory and I/O costs. Potentially, it requires large amounts of memory to hold the in-memory data structures, and thus it may not be suitable for a dataset with a very large number of merger events. The I/O cost of scanning the entire table can be amortized over queries for multiple merger trees.

4.4 Approach 3: In-Memory Forest Queries

This approach is a modification of the In-Memory Queries one. The basic idea is that instead of building the merger tree for a set of query `qbhids`, the complete merger forest is built upon scanning the the ME table. Although this approach incurs extra work to build all the trees, this cost is amortized when a large number of queries need to be processed. This approach is based on the Union-Find algorithm [14] and adjusted to handle the peculiarities of the merger events representation, such as the fact that there is no explicit link that indicates the relationship between an ME node and its left child (bh1). The procedure builds the tree structure for each of the connected components.

This approach uses two associated set structures (e.g., hash tables or dictionaries). The first one, bh1Map maps from bh1 to the list of merger events that share the same value of bh1. The second one, bh2Map maps from bh2 to a single ME record that has the appropriate bh2 value. As the records are scanned from the database, they are added to bh1Map and bh2Map as shown in the BuildForestInMemory procedure. Then, the right-side links are created by iterating over the bh2Map and searching for the corresponding list of nodes in bh1Map, i.e., it creates a link where node.bh2 = node1.id. The left-side links are created by iterating over the lists found in bh1Map. During this loop, the root nodes for all the trees are determined using the findRootAndAddToForest procedure. This procedure only adds new trees to the forest and returns early when the root for a tree has already been found. Finally, the BuildForestInMemory procedure returns a tuple containing the forest and the maps from the ids to the tree nodes. Queries are processed by looking up the desired qbhid in either bh1Map or bh2Map to obtain a starting node in the tree from which the root of the tree can be obtained. From the root, all the nodes in the tree can be returned.

```
Procedure BuildForestInMemory(db):
Input:  the DB with the ME table
Output: a forest containing all the merge trees in ME

  // Scan over all ME records
  cursor = SELECT bh1, bh2, time FROM ME;

  foreach (bh1, bh2, time) in cursor
    node = new TreeNode(bh1, time, bh2)
    bh2Map.put(bh2, node)       // Map from bh2 to this node
    bh1Map.addToList(bh1, node) // Map from bh1 to a node list

  foreach node in bh2Map
    // Create the link for the right-side child
    node.right = bh1Map.get(node.bh2)  // It may be null

  forest = emptySet()
  foreach lst in bh1Map
    sortbytime(lst)
    // Create links from lst[n-1].left to lst[n]
    createLinkOnBh1(lst)
    findRootAndAddToForest(lst, forest)

  return (forest, bh1Map, bh2Map)
```

4.5 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. 4), 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 built a prototype implementation of 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.

5.1 Workload

The dataset was produced by a ΛCDM cosmology simulation using the GADGET-3 [28] parallel program. The simulation ran at the National Institute for Computational Sciences (NICS) using all 99072 processors of the Kraken supercomputer. The simulation contained 33 billion dark matter particles and 33 billion hydro particles in a box of $533.33h^{-1}$Mpc in size. A snapshot of all the particles is 3 TB in size. At the end of the simulation ($z = 4.7$), 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 5 shows the distribution of tree sizes in number of merger events in the ME table.

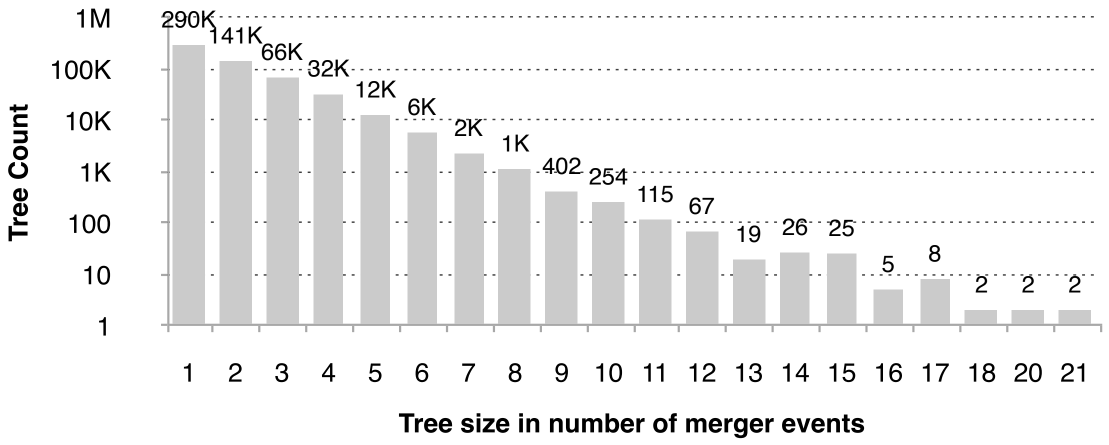


Figure 5: Distribution of tree sizes in the black holes dataset. The X axis is the size of a merger tree measured as the number of events in a tree. The Y axis is the number of trees of that size in $\log_{10}$ scale.

5.2 Storage Requirements

For our prototype implementation, we stored the BH data in SQLite (v3.7.3). We gathered the storage requirement of storing the BH dataset in the database and compared it to a raw binary representation. In the binary layout, the records are stored in a dense array of size $N \times \mathrm{sz}$, where N is the number of records in the table and sz is the size of an equivalent unpadded C structure. Table 1 shows the sizes of the main tables in SQLite and the corresponding size of the binary array representation. We built various indices required for speeding up the queries to the BH and ME tables. Table 2 contains the size of the indices and additional summary and provenance database tables.

Table 1: Size of main tables in the BH dataset

Table	- Records	- Tab. Size	- Bin. Size
BH	420 M	50 GB	22 GB
ME	1 M	49 MB	26 MB
NM	175 M	13 GB	4.6 GB
Total		63 GB	27 GB

Table 2: Size of indices and auxiliary tables

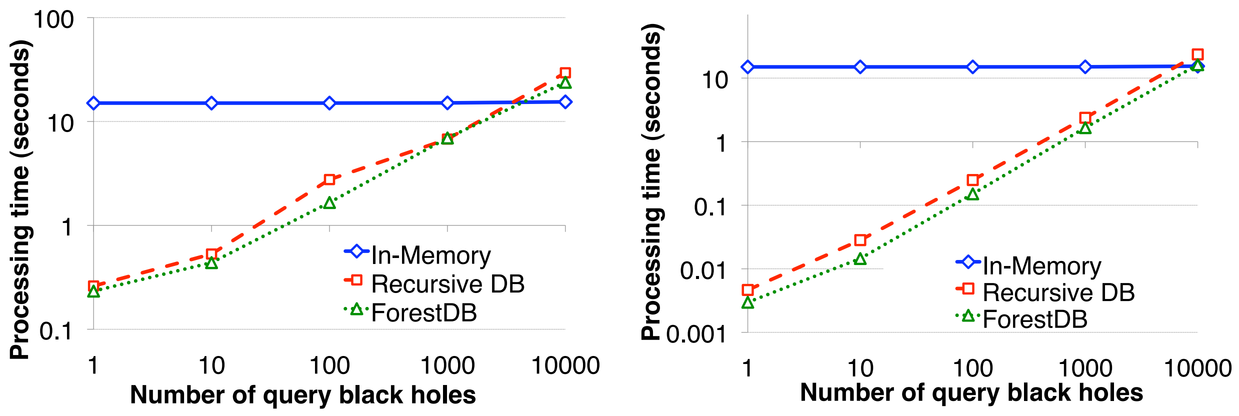
Item	Size
BH index on bhid	8 GB
BH index on time	8 GB
ME index on bh1	17 MB
ME index on bh2	15 MB
MF table	53 MB
MF indices bh1,bh2	32 MB
MF index on tid	12 MB
Aux. and provenance data	6 GB

5.3 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.

Setup. The experiments were run on a server host with 2 GHz dual core Intel(R) Xeon(R) CPUs, 24 GB of memory and a 0.5 TB software RAID 1 volume over two SATA disks. The OS was Linux(R) running a 2.6.32 kernel. Our prototype implementation of the different approaches is written in Python (v3.1) and the data is stored in SQLite.

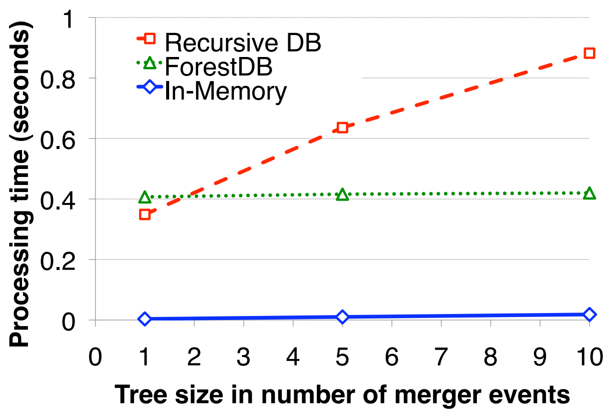
Building Merger Trees. The first set of micro benchmark experiments corresponds to the steps needed to build the merger trees for a set of query black holes (qbhs). We compared three of the approaches explained in Sec. 4: (a) Recursive DB – RDB, (b) In-memory – IM, and (c) Forest DB – FDB. The In-memory Forest approach (Sec. 4.4) 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.



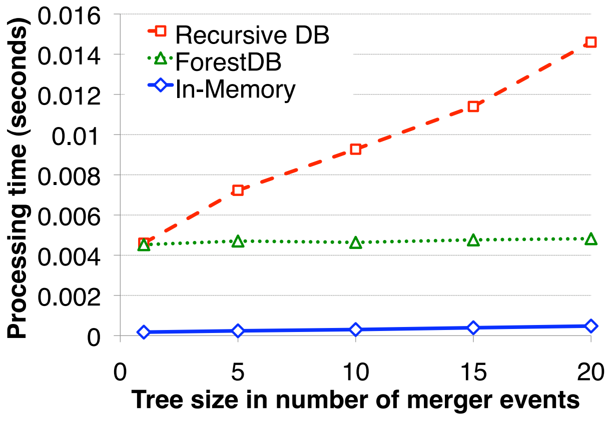
(a) Cold OS cache (b) Warm OS cache

Figure 6: 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. 6. 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. 6a), 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.



(a) Batch size=250 qbhs

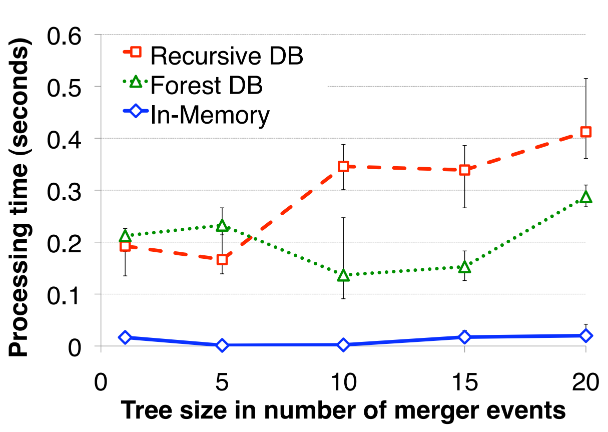


(b) Batch size=2 qbhs

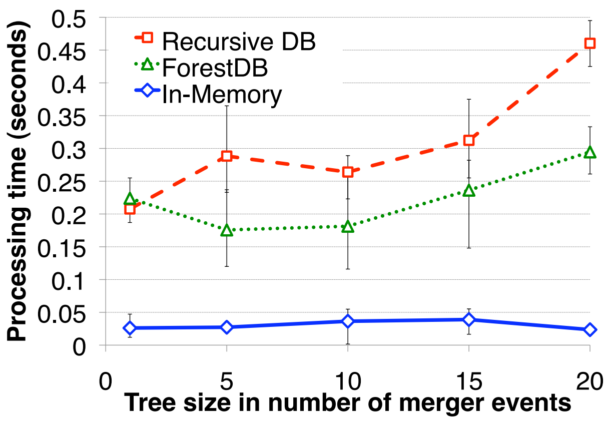
Figure 7: Processing time for building the merger trees using various approaches. This experiment was performed with a warm OS cache and a cold DB cache. The X axis is the size of the resulting tree; (a) and (b) show the time to process 250 qbhs and 2 qbhs per request respectively. The Y axis is the elapsed time to build the number of trees of each size.

Figure 7 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.

Figure 8 shows the processing time according to tree size for requests of size 2 qbhs and cold OS cache. In these experiments, a request for 2 qbhs (of a given tree size) is repeated 9 times with cold cache. The minimum, average. and maximum are shown for each point. Figure 8a shows the result for a particular set of request qbhs, and Fig. 8b shows the aggregate across different 2-qbh request sets. The difference between the RDB and FDB approaches is inconclusive in this scenario, especially for small trees. In this case, the request time is dominated by the characteristics of the



(a) Request size = 2 qbhs



(b) Average over requests for 2 qbhs

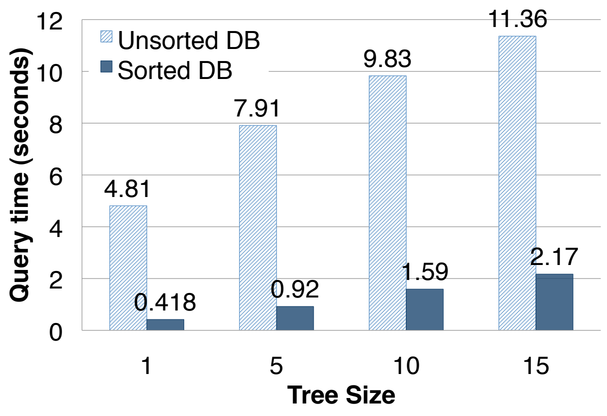
Figure 8: Processing time for building the merger trees for 2 qbhs per request with cold OS and DB caches. The X axis corresponds to the tree size. The Y axis is the elapsed time in seconds to build the trees. (a) shows the time for a particular request set with 2 qbhs. The average processing time for multiple 2-qbhs sets is shown in (b).

I/O devices, rather than the particular approach.

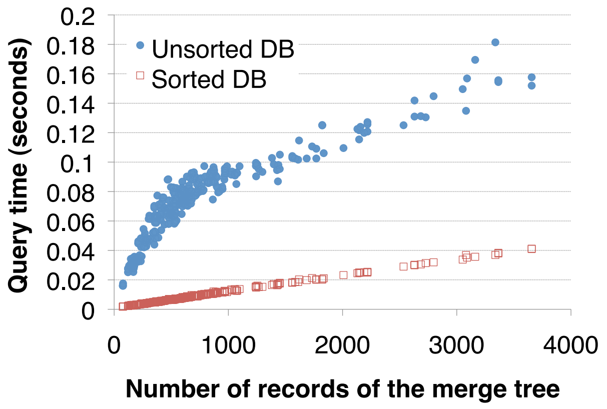
Retrieving the Time History for Merger Trees. In the second set of experiments, we retrieved the detailed time history for a set of trees retrieved in the previous step. This entails retrieving from the BH table all the records for the corresponding BH in a given merger tree. For each tree size (1, 5, 10, 15), we retrieved the BH histories for 100 trees of that size. Figure 9a 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 9b 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.

5.4 Pre-processing

Creating the BH database involves loading the data and creating the needed indices. Although this is a one-time cost, it is a necessary step to enable the data analysis. If this process takes too long, it may become a barrier for adoption of the DB approach. The initial loading operation presents a good opportunity for performing needed data cleanup operations such as removing duplicate or unwanted records. Loading the database takes 10 hours using our pre-processing scripts in our platform. Under closer examination we determined that the process was CPU intensive. Its running



(a)



(b)

Figure 9: 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.

time can be greatly reduced with a compiled language implementation and by processing in parallel subsets of the data being loaded. Creating the MF table takes 55–65 *s* once the data is loaded. The pre-processing time of the current implementation is already a significant improvement over the previous custom approach where creating the binary files required days. In the custom approach, the data of recent simulations could not be loaded into a single file in a single step. Instead, multiple binary files need to be created in order to deal with the larger data sizes. For example, creating the binary file for the last fraction of the most recent simulation took approximately 12 days. The resulting files cannot be easily queried at once. In contrast, with the DB approach, not only the initial pre-processing time is lower, but also all the data is in a single database that can be queried in a consistent and flexible manner.

6 Future Work

Now that scientists are able to easily query the data, they are able to carry more involved and extensive types of analysis, which in turn results in more complex queries and access patterns. The next step is to add other types of data, such as information about galaxy halos, in order to correlate black holes with their surrounding environment. Adding new types of data brings challenges due to the size of those other data sources. Current black hole datasets can be managed with RDBMS using a single server-class host. The size of these datasets will increase as cosmology simulations grow larger. Alternative, more efficient approaches will be needed to manage and analyze these coming datasets. We are developing hybrid array-database and compression techniques for storing black holes histories using more compact representations. To address the challenges of scaling to larger data sizes, we are developing a new generation of tools that will leverage distributed, scalable, structured table storage systems such as HBase and Cassandra. Currently, the data analysis is delayed by the time required to load the simulation output into the database. Although, it is a one time cost, the time required to load multi-terabyte and petabyte datasets is potentially long, in the order of multiple days on relatively small computer clusters. We are exploring efficient *in-situ* processing and bulk loading mechanisms to address this issue.

7 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.

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 Parallel Data Laboratory Consortium (PDL) at Carnegie Mellon University (CMU). The computations were supported by an allocation of advanced computing resources supported by the NSF and the TeraGrid Advanced Support Program. The computations were performed on on Kraken at the National Institute for Computational Sciences (NICS) http://www.nics.tennessee.edu and the CMU Open Cloud Cluster http://wiki.pdl.cmu.edu/opencloudwiki

References

[1] ABAZAJIAN, ET AL. 7-th Data Release of the Sloan Digital Sky Survey. *ApJS 182* (2009).

[2] BAUMANN, P., FURTADO, P., RITSCH, R., AND WIDMANN, N. Geo/environmental and medical data management in the rasdaman system. In *23th VLDB Conf.* (1997), pp. 548–552.

[3] BAYER, R., AND MCCREIGHT, E. M. Organization and maintenance of large ordered indices. In *Proc. of SIGFIDET Workshop on Data Description and Access* (Rice University, Houston, TX, USA, Nov 15-16 1970), ACM, pp. 107–141.

[4] BRUNNER, R. J., GRAY, J., KUNSZT, P., SLUTZ, D., SZALAY, A. S., AND THAKAR, A. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. Tech. Rep. MSR-TR-99-30, Microsoft Research, June 1999.

[5] COHEN, S., AND GUZMAN, D. E. Sql.ct: Providing data management for visual exploration of ct datasets. In *Proc. Int. Conf. on Scientific and Statistical Database Management (SSDBM)* (2006).

[6] 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.

[7] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. In *Symp. on Operating System Design and Implementation (OSDI)* (San Francisco, CA, Dec 2004).

[8] DEGRAF, C., DI MATTEO, T., AND SPRINGEL, V. Faint-end quasar luminosity functions from cosmological hydrodynamic simulations. *Monthly Notices of the Royal Astronomical Society (NMRAS) 402* (Mar. 2010), 1927–1936.

[9] DEGRAF, C., DI MATTEO, T., AND SPRINGEL, V. Quasar Clustering in Cosmological Hydrodynamic Simulations: Evidence for mergers. *ArXiv e-prints* (May 2010).

[10] DI MATTEO, T., COLBERG, J., SPRINGEL, V., HERNQUIST, L., AND SIJACKI, D. Direct Cosmological Simulations of the Growth of Black Holes and Galaxies. *Astrophysical Journal (ApJ) 676* (Mar. 2008), 33–53.

[11] DI MATTEO, T., SPRINGEL, V., AND HERNQUIST, L. Energy input from quasars regulates the growth and activity of black holes and their host galaxies. *Nature 433* (Feb. 2005), 604–607.

[12] FOLK, M., CHENG, A., AND YATES, K. Hdf5: A file format and i/o library for high performance computing applications. In *Proc. of Supercomputing (SC)* (1999).

[13] FU, B., REN, K., LÓPEZ, J., FINK, E., AND GIBSON, G. Discfinder: A data-intensive scalable cluster finder for astrophysics. In *Proc. Int. Symp. on High Performance Distributed Computing (HPDC)* (Chicago, IL, June 2010), IEEE/ACM.

[14] GALLER, B., AND FISCHER, M. An improved equivalence algorithm. *Communications of the ACM (CACM) 7*, 5 (1964), 301–303.

[15] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In *Proc. Intl. Conf. on Management of Data* (Boston, MA, Jun 1984), Association of Computing Machinery (ACM), Special Interest Group on Management of Data (SIGMOD), pp. 47–57.

[16] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the European Conference on Computer Systems (EuroSys)* (2007), pp. 59–72.

[17] IVANOVA, M., NES, N., GONCALVES, R., AND KERSTEN, M. Monetdb/sql meets sky-server: the challenges of a scientific database. In *Proc. of the 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)* (2007), p. 13.

[18] KAUSTAV DAS, J. S., AND NEILL, D. Anomaly pattern detection in categorical datasets. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2008)* (August 2008).

[19] KWON, Y., NUNLEY, D., GARDNER, J. P., BALAZINSKA, M., HOWE, B., AND LOEBMAN, S. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. In *Proceedings of the 22nd Scientific and Statistical Database Management (SSDBM)* (2010).

[20] LEE, D., AND WONG, C. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9 (1977), 23–29.

[21] LEMSON, G., AND SPRINGEL, V. Cosmological simulations in a relational database: Modelling and storing merger trees. In *Astronomical Data Analysis Software and Systems* (2006), no. XV in ASP Conference Series, p. 212.

[22] LEPENDU, P., DOU, D., FRISHKOFF, G. A., AND RONG, J. Ontology database: A new method for semantic modeling and an application to brainwave data. In *Scientific and Statistical Database Management* (2008), vol. 5069, pp. 313–330.

[23] LOEBMAN, S., NUNLEY, D., KWON, Y., HOWE, B., BALAZINSKA, M., AND GARDNER, J. P. Analyzing massive astrophysical datasets: Can pig/hadoop or a relational dbms help? In *Proc. IASDS* (2009).

[24] OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R., AND TOMKINS, A. Pig latin: A not-so-foreign languge for data processing. In *Proceedings of the SIGMOD Conference* (2008), pp. 1099–1110.

[25] REW, R. K., AND DAVIS, G. P. The unidata netcdf: Software for scientific data access. In *Proc. of the Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology* (1990), pp. 33–40.

[26] SAMET, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[27] SCHLOSSER, S. W., RYAN, M. P., TABORDA, R., LÓPEZ, J., O'HALLARON, D. R., AND BIELAK, J. Materialized community ground models for large-scale earthquake simulation. In *Proc. Conference on Supercomputing (SC'08)* (Piscataway, NJ, USA, 2008), ACM/IEEE, IEEE Press, pp. 1–12.

[28] SPRINGEL, V. The cosmological simulation code GADGET-2. *Monthly Notices of the Royal Astronomical Society (NMRAS) 364* (Dec. 2005), 1105–1134.

[29] SPRINGEL, V., DI MATTEO, T., AND HERNQUIST, L. Modelling feedback from stars and black holes in galaxy mergers. *Monthly Notices of the Royal Astronomical Society (NMRAS) 361* (Aug. 2005), 776–794.

[30] SPRINGEL, V., AND HERNQUIST, L. Cosmological smoothed particle hydrodynamics simulations: the entropy equation. *Monthly Notices of the Royal Astronomical Society (NMRAS) 333* (July 2002), 649–664.

[31] SPRINGEL, V., AND HERNQUIST, L. Cosmological smoothed particle hydrodynamics simulations: a hybrid multiphase model for star formation. *Monthly Notices of the Royal Astronomical Society (NMRAS) 339* (Feb. 2003), 289–311.

[32] SZALAY, A., BUDAVARI, T., CONNOLLY, A., GRAY, J., MATSUBARA, T., POPE, A., AND SZAPUDI, I. Spatial clustering of galaxies in large datasets. *Astronomical Data Analysis II 4847*, 1 (2002), 1–12.

[33] TARJAN, R. E. Efficiency of a good but not linear set union algorithm. *J. ACM 22*, 2 (1975).

[34] TU, T., LÓPEZ, J., AND O'HALLARON, D. The Etree library: A system for manipulating large octrees on disk. Tech. Rep. CMU-CS-03-174, Carnegie Mellon School of Computer Science, Pittsburgh, PA, July 2003.

[35] TU, T., O'HALLARON, D., AND LÓPEZ, J. Etree – a database-oriented method for generating large octree meshes. In *Proceedings of the Eleventh International Meshing Roundtable* (Ithaca, NY, Sep 2002), pp. 127– 138.

[36] WELLS, D. C., GREISEN, E. W., AND HARTEN, R. H. Fits - a flexible image transport system. *ASTRONOMY AND ASTROPHYSICS. SUPP. SER. 44* (1981), 363.

[37] XU, W., OZER, S., AND GUTELL, R. R. Covariant evolutionary event analysis for base interaction prediction using a relational database management system for RNA. In *Proc. 21th Conf. on Scientific and Statistical Database Management (SSDBM)* (2009).

[38] YANG, Y.-S., HSIEH, S.-H., TSAI, K.-C., WANG, S.-J., WANG, K.-J., CHENG, W.-C., AND HSU, C.-W. Isee: Internet-based simulation for earthquake engineering - part i: Database approach. *Earthquake Engineering & Structural Dynamics 36* (2007), 2291–2306.