Efficient evaluation of threshold queries of derived fields in a numerical simulation database

Kalin Kanov
Department of Computer Science
IDIES
Johns Hopkins University
Baltimore, MD 21218
kalin@cs.jhu.edu

Randal Burns
Department of Computer Science
IDIES
Johns Hopkins University
Baltimore, MD 21218
randal@cs.jhu.edu

Cristian C. Lalescu
Department of Applied Mathematics and Statistics
IDIES
Johns Hopkins University
Baltimore, MD 21218
clalesc1@jhu.edu

ABSTRACT

In this paper, we present a method for the efficient evaluation of threshold queries of derived fields for large numerical simulation datasets stored in a cluster of relational databases. The datasets produced by these simulations are in the TB and even PB ranges. Data-intensive computations that examine entire time-steps of the simulation data are impractical to perform locally by the user, taking days or months to iterate over the entire dataset. The integrated method for the evaluation of threshold queries that we have developed achieves scalability through data-parallel execution of the computations on the nodes of an analysis database cluster. We extend the scientific analysis environment with the introduction of an application-aware cache for query results, building on the concept of semantic caching. The cache has little overhead and improves query performance by over an order of magnitude for queries that hit the cache. Caching the results of threshold queries preserves both the I/O and computation effort used to obtain them. In the case of computational turbulence, this allows scientists to quickly focus on the most intense events and interesting regions in any time-step or the dataset as a whole, which greatly speeds up the rate of scientific exploration and discovery.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – Scientific Databases; H.2.4 [**Database Management**]: Systems – Distributed Databases; J.2 [**Computer Applications**]: Physical Sciences and Engineering – Physics

Keywords

scientific databases, data-intensive computing, threshold queries, turbulence

1. INTRODUCTION

Better instruments, faster and bigger supercomputers and easier collaboration and sharing of data in the sciences have introduced the need to manage increasingly large datasets. Data-intensive systems and architectures have been developed with the goal of storing and providing fast access to such datasets. Examples of such analysis environments include the GrayWulf and Data-Scope clusters [31, 10] at Johns Hopkins, which have capacity of 1.1PB and 11PB respectively. One of their missions is to provide persistent storage and public access to world-class numerical simulation data. These systems differ from the traditional HPC environments in that they aim to achieve high aggregate throughput by balancing computation capabilities with I/O and network bandwidth. The computing systems and services developed on top of these platforms are more than pure storage engines and usually have complex analysis routines built-in, which has largely been driven by the "move the computation to the data" paradigm [14]. These built-in analysis routines are most often not novel themselves. They implement core scientific functionality for the study of the particular scientific phenomena, which was observed or simulated in the first place. The analysis routines however require novel evaluation strategies and methods for their execution. They have to operate on large array datasets distributed across multiple nodes of a cluster of relational databases. In order to reduce their running times, they have to make efficient use of the cluster resources and incorporate leading data management techniques.

Finding the locations or regions of highest vorticity or those with the largest norms of the velocity or other fields of interest enables new insights in the study of fluid dynamics. Analysis of this kind coupled with the ability to analyze time-series datasets both forward and backward in time has transformed our understanding of turbulence [12]. Furthermore, threshold, top-k queries and similarity search in general are important in many different disciplines. We introduce an efficient evaluation strategy for threshold queries over time-series datasets stored in a cluster of relational databases. Our method evaluates not only threshold queries of the vector or scalar field data stored in the database, but also performs thresholding of *derived* fields.

The main challenge that our approach tackles is that the data-intensive computation of *derived* fields has to be carried out on-demand for extremely large array datasets stored in an analysis cluster environment comprised of multiple

10.5441/002/edbt.2015.27

database nodes. We focus on the evaluation of threshold queries of fields derived from the data stored in the cluster as these queries are the most interesting scientifically. However, our approach applies to the evaluation of top-k queries, rollup queries and data-reducing queries in general. The established data management techniques that our approach combines make the approach easy to understand. It can be applied to other scientific analysis environments, which manage large datasets in a database management system. Examples include the Sloan Digital Sky Survey [28], the Millennium Simulation [23] and the Open Connectome Project [6].

Evaluating threshold queries within the database cluster allows scientists with modest computational and network capability to narrow down on and examine some of the most interesting regions and features in the dataset and focus on the subsequent analysis needed to understand these events. It is impractical to materialize all possible derived fields and store them alongside the raw data due to the large size of the datasets and the limits of available storage. Obtaining the derived field and thresholding locally by the user requires not only the computation of the derived field over an entire time-step server-side but also the transfer of a large amount of data over the network, most of which are subsequently discarded. One of our collaborators reported that such a local evaluation of a threshold query over an entire time-step took over 20 hours. It would take months to iterate over the entire dataset. This highlighted the need for providing the capability through an integrated approach, which performs the evaluation server-side.

Database, operating and file system caches are effective at speeding up access to the large amounts of data stored on disk. However, this might not be sufficient for some applications, because these application-independent caches cannot exploit dataset-specific structure and application-level information [20]. Moreover, even if the data are available in one or more of these application-independent caches the computation associated with the derived field still needs to be performed for each point on the grid, because results of previous computations are not cached. We will demonstrate that an integrated approach, which computes the derived fields on-demand in a data-parallel manner, performs the evaluation over an entire time-step in a few minutes. Storing the query results in an application-aware semantic cache further reduces the running times to several seconds.

Thresholding allows scientists to obtain and examine the regions containing the most intense events and features in the dataset in the case of turbulence. These are often the locations that have the largest vorticity norms and have intense vortices or reconnection events. In magnetohydrodynamics, the locations of largest electric current are of great interest for similar reasons. It is important that threshold queries are evaluated in an efficient manner, because often further subsequent examination and analysis is required to understand the physics that drive these intense events.

There are several challenges that arise during the evaluation of threshold queries of derived fields in an analysis database cluster. The field variables have to be evaluated *on-demand* from the array data stored in the database cluster. The evaluations are data-intensive as they perform kernel computations on extremely large multidimensional array datasets. A kernel computation computes the value at a grid location using the data points at a set of neighboring locations. Kernel computations have to be performed at each location on the grid as opposed to at a particular number of target locations. The evaluation needs to be distributed across the nodes of the database cluster to avoid the unnecessary movement of data over the network and to achieve scalability. Techniques that target the traditional supercomputing environments do not translate directly to the distributed database setting of an analysis cluster environment.

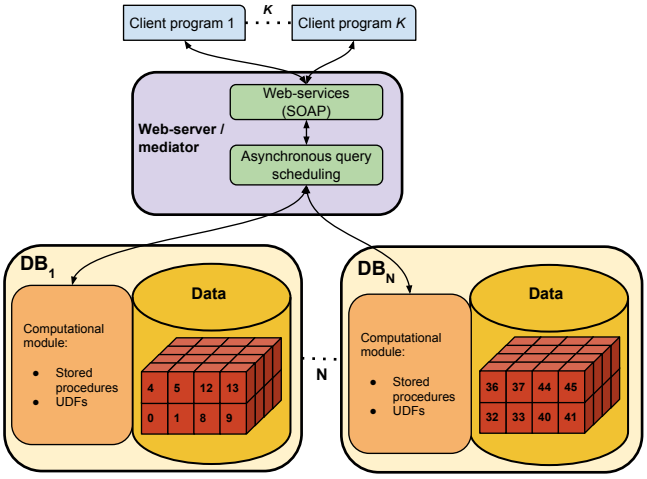


Figure 1: Architecture of the JHTDB.

We present a method for the efficient evaluation of threshold queries over fields derived from the raw vector or scalar fields of the numerical simulation stored in the database. Our method makes effective use of the cluster resources and achieves high throughput and scalability. We exploit the parallelism available in the cluster by means of data-parallel execution of the computations. We extend the database management system with an application-aware cache for query results. We build on the idea of an application-aware cache introduced by Lopez et al. [20] and more broadly on the concept of semantic caching [9]. Rather than caching just data as is the case in system caches and the tree-cache described by Lopez et al., we cache query results along with query metadata and subsequent queries are evaluated against the cache. This leads to query performance improvement of over an order of magnitude.

The contributions of this paper are the following:

- Computing derived fields of large simulation data on-demand and evaluating threshold queries on them at extreme scale. This provides large data analytics capabilities that examine entire time-steps of the simulation transparently to the user in a production analysis environment.
- Achieving this through the combination of existing data management techniques such as data parallelism and semantic caching as well as taking advantage of heterogeneous scientific cluster architectures (sharded relational DBMS with several SSDs per node).
- Evaluating the proposed method on data-intensive workloads in a live production environment and showing scalability results on datasets hundreds of terabytes in size.

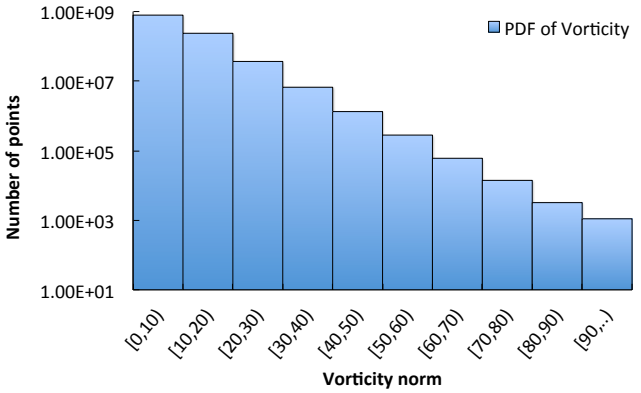


Figure 2: Probability density function of the norm of the vorticity field for a representative time-step for the MHD dataset.

2. JOHNS HOPKINS TURBULENCE DATABASES

Data-intensive architectures and compute clusters built from commodity hardware rely on parallel I/O to multiple disks and high network bandwidth to achieve high throughput. Such systems have only recently been deployed for the storage of large numerical simulation datasets. The virtual laboratories built on these systems make use of relational database system technology to store and manage large array datasets. Relational database systems however often do not support all of the functionality that scientists are interested in out of the box. It is either up to the user to develop more sophisticated analysis routines locally or such capabilities have to be built into the database through user-defined functions or stored procedures.

The method that we have developed for the evaluation of threshold queries of derived fields was deployed and integrated into the Johns Hopkins Turbulence Databases (JHTDB) [19, 26]. It solves a pressing problem in a production scientific analysis environment, which differs from the traditional supercomputing environments and provides large data analytics capabilities transparently to the public. The JHTDB, built on top of the GrayWulf and Data-Scope clusters, serves as a public virtual laboratory for the study of turbulent phenomena. The JHTDB stores several datasets, which are the output of high-resolution numerical simulations of turbulence. The 3d time-series data are partitioned into small subcubes and stored in relational databases distributed across the nodes of the cluster. Access to the data is provided by means of Web-services and a variety of analysis functions have been implemented and can be executed through Web-service calls (Fig. 1). At present the service hosts four datasets, which are available publicly. The data are the output of numerical simulations of forced isotropic turbulence, magnetohydrodynamics (MHD), channel flow turbulence and homogenous buoyancy driven turbulence. The total amount of space occupied by the datasets is over 230 TB.

The database nodes are part of the GrayWulf [31] and Data-Scope [10] clusters. Each node is running Windows Server 2008 and SQL Server 2008 R2. The data for each dataset reside on a regular three-dimensional spatial grid with the exception of the channel flow data, which has an irregular y dimension. The data are partitioned spatially across 4 to 8 database nodes, and each database node hosts one or more databases. We use the Morton z-order space-filling curve to distribute the data across nodes and databases [26]. Each time-step is spatially subdivided into database atoms, which are of size 8^3. Each such atom is indexed by the time-step, which it belongs to and by the Morton code of it's lower left corner. This combination of index and data forms a record in the database. Queries to the data and derived fields, such as derivatives and filtered quantities are evaluated through stored procedures or user-defined functions implemented in the Common Language Runtime (CLR) framework.

The Web-services are hosted on a front-end Web-server, which handles user requests and hosts the main Web-page portal. The Web-server acts as a mediator sending the users' requests to the database nodes and initiating their distributed evaluation. Each request is broken down into multiple parts based on the spatial layout of the data. Each part is asynchronously submitted for evaluation to the database which stores the data needed for the evaluation. The Web-server assembles the results from the distributed computation and sends them back to the client.

The JHTDB provides a variety of data-intensive analysis routines that are executed on the database nodes. These include interpolation, differentiation, particle tracking and spatial filtering. These tasks are often data-intensive and in order to leverage the capabilities of the cluster we have developed data-driven batch processing techniques for their evaluation [17, 16]. Most of these tasks usually operate on subsets of the space or a collection of individual target locations within a time-step.

In contrast, threshold and top-k queries usually have to examine the entire data volume of a time-step or a signification portion of it. Furthermore, the data product of threshold queries is much smaller in relation to the amount of data that need to be examined. This fact combined with the fact that subsequent queries can reuse previously computed results makes the query results suitable to caching.

3. SCIENTIFIC USE CASES

One of the applications of thresholding in turbulence is to find the locations of maximum vorticity in a particular time-step or the dataset as a whole. The locations of maximum vorticity are usually associated with the most intense vortices in the dataset and often have interesting and complex reconnection events associated with them. Once obtained from the service, these locations can be clustered in both 3d and 4d. This allows scientists to examine their evolution with the flow and make subsequent analysis queries as needed in order to study these events. The relationship between different "worms" (see Figure 3) that connect and reconnect at those locations is of the most interest.

The vorticity is computed from the velocity field by taking its curl:

$$\vec{\omega} = \vec{\nabla} \times \vec{v} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \times (v_x, v_y, v_z)$$
$$= \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right). \quad (1)$$

We use finite differencing methods of different orders for the evaluation of the curl. For example, with 4th-order centered

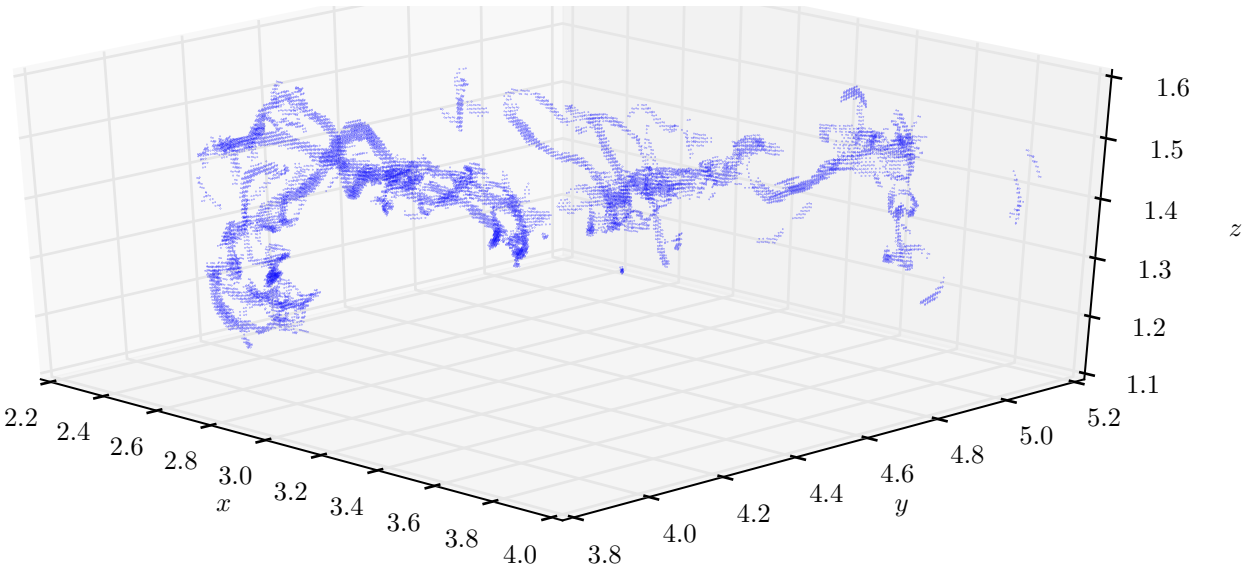


Figure 3: 3D (single time-step) cut through the 4D cluster containing the most intense event.

finite differencing each partial derivative is evaluated from the 4 adjacent grid node values as follows:

$$\left.\frac{df}{dx}\right|_{x_n} = \frac{2}{3\Delta x}[f(x_{n+1}) - f(x_{n-1})] - \frac{1}{12\Delta x}[f(x_{n+2}) - f(x_{n-2})], \quad (2)$$

where f denotes any one of the three components of the velocity and Δx is the width of the grid in the x direction. The partial derivatives along y and z are computed in the same fashion. Figure 2 shows the distribution of the values of the norm of the vorticity field in the MHD dataset for a representative time-step. This is indicative of how the values are distributed in the dataset as a whole. This coarse view of the data can be used by scientists to guide the selection of threshold values.

Figure 3 shows the most intense event observed in the forced isotropic turbulence dataset. The locations of maximum vorticity in the dataset were clustered in this case in 4d using a friends-of-friends algorithm. It is interesting to note that the cluster containing the most intense event in the entire dataset develops from nothing (i.e. it does not appear in the first few time-steps) and it takes less than the timespan stored in the database for it to develop. Figure 3 also shows that most interactions between worms are not simple. There are several worms interacting in a complex way at the same time. Similar type of analysis and the fact that the entire time history of the simulation is available in a database cluster, which provides built-in sophisticated analysis routines revealed flux-freezing breakdown in MHD turbulence [12], showing why solar flares last minutes rather than the millions of years that conventional theory would predict.

In addition to obtaining the regions of largest vorticity, there is substantial interest in studying the regions with highest values for other fields, such as the second and third velocity gradient invariants (Q and R). These invariants are scalar quantities whose values contain information about the topology of the flow and the rates of vortex stretching and rotation. In MHD, finding the locations with largest values for the electric current can lead to new insights into the development of the most intense reconnection events of magnetic field sheets in the simulated plasma. Similarly to the vorticity, the electric current is derived from the magnetic field by taking its curl. The list of fields of interest, on which scientists would like to perform threshold queries certainly does not stop here and is indicative of how valuable this functionality is in the study of turbulence and fluid dynamics.

4. THRESHOLD QUERY EVALUATION

Threshold queries of derived fields submitted to the JHTDB are evaluated using a data-parallel execution strategy and the query results are cached in an application-aware semantic cache. In addition to query results, the cache stores their semantic descriptions and query metadata and parameters used to obtain them. The evaluation strategy for queries that do not hit the cache is driven by the spatial partitioning of the data across the nodes of the cluster.

Derived fields computation: The databases store only the raw field data from the simulation (e.g. velocity, pressure, magnetic field etc.). However, the threshold queries of most interest to science users produce all locations where the values of a *derived* field are above a given threshold. Thus, the derived field in question has to be computed from the raw data first. For most derived fields of interest, this computation has local support. It has an associated localized kernel of computation around each grid node. Therefore, the value of the derived field at each grid node depends on the value of the stored field at all of the grid locations, which are part of the kernel of computation.

Distributed data-parallel execution: In most cases threshold queries operate over an entire time-step. Each such query is subdivided by the mediator into queries submitted to each of the database nodes. Each node evaluates the query over the data that it has stored locally. Only a

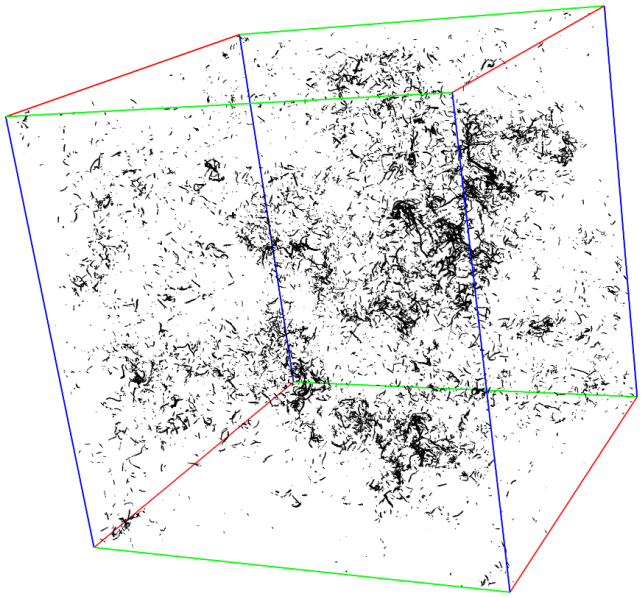


Figure 4: Points with values above 7 times the root mean square value of the vorticity for a single time-step.

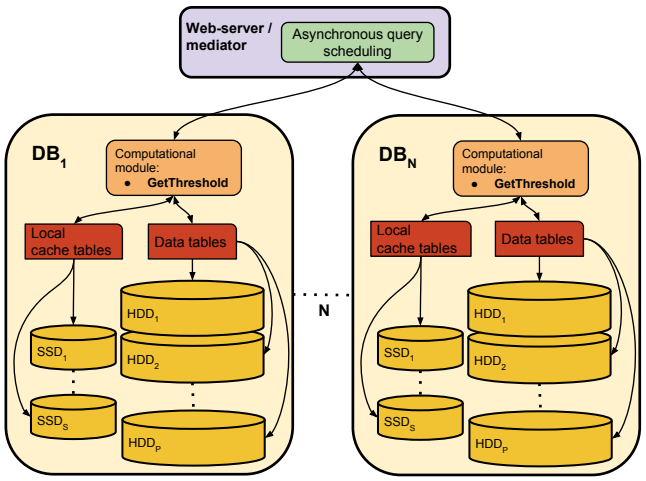


Figure 5: Distributed evaluation of threshold queries and architecture of the application-aware cache. Each database node has local cache tables, which reside on solid-state drives attached to the node.

small amount of data along the boundary need to be requested from adjacent nodes. The size of the band of data that may not be available locally is equal to a kernel half-width. Such a band is needed on each of the sides of the box forming the domain of the computation. The data are read into memory and the particular field requested is computed at each of the locations on the grid. The same strategy applies when utilizing multiple processes per node. The norm or absolute value of the derived field at each location is compared against the specified threshold and if the value is higher, it is maintained along with the spatial coordinates of the location in a list.

We impose a limit on the maximum number of locations that can be returned as a result of a threshold query in order to prevent having to return the entire time-step or a significant fraction of it for queries with thresholds that are set too low. Currently this limit is set conservatively to 10^6 locations, which is sufficient to examine a time-step in detail. In the case of the vorticity, the values above 8 times the root mean square value, which is about 25% of the maximum, are contained within 2.6×10^5 points in each time-step. Figure 4 shows all the points in a single time-step with values above 7 times the root mean square value. There are 2.4×10^5 points in the figure. Given that we are interested in extreme events, obtaining the locations with values even within 50% of the maximum would be sufficient. At the same time this also limits the amount of data that have to be returned to the user over the network as well as the amount of data that have to be cached. Users receive an error message notifying them if their request has a threshold that is set too low. If a user is interested in obtaining more data he or she can request the values of the derived field directly. Alternatively, if they are interested in the density distribution of values they can examine the probability density function (e.g. Fig. 2), which is computed using a similar strategy to threshold queries.

Application-aware cache for query results: A central part of the evaluation strategy for threshold queries that we have developed is the introduction of an application-aware cache for query results (Fig. 5). The results of these queries are small compared to the amount of data that need to be examined and the results can be used to answer subsequent queries as long as they are within the same region and specify the same or higher threshold. Each database node has a local cache. Cache entries are indexed by the field, time-step, spatial region and the threshold requested. We use a least recently used cache replacement policy. All modifications of and queries to the cache are executed within a transaction with snapshot isolation level to avoid dirty-reads or an inconsistent view of the cache.

Caching the query results preserves the computational effort in addition to substantially reducing I/O. The cached data are for the particular derived field that was queried and not the raw data of the simulation fields. Thus, we do not have to derive the requested field from the raw data for queries that hit the cache. This results in a substantial improvement in query performance as we only have to scan a small set of data and do not need to perform any additional computation.

Not all query results are suitable to caching. Most of the queries submitted to the JHTDB other than threshold queries request data at a collection of target locations. Given that there are 1024^4 possible locations for three of the datasets and $6 * 1024^4$ locations for the channel-flow dataset the chance of reuse for the results of these queries is extremely small. This is why the cache currently stores only the results of threshold queries. Nevertheless, it can easily be extended to cache the results of other query types as well if that becomes advantageous.

The cached query results are stored in a table in the database and the overall size of the cache is limited by the amount of available SSD disk space, not memory. Given a limit of $\sim 10^6$ points per time-step for a threshold query, the space required to cache the maximum number of points in-

cluding the index space and database overhead is ∼40MB. Therefore for a dataset containing 1024 time-steps, as is the case for the isotropic turbulence and MHD datasets part of the JHTDB, a cache size of 40GB is sufficient to cache the query results for threshold queries of a derived field over the entire dataset. The currently available SSD disk space per node is ∼200GB, which will be sufficient to maintain the threshold results for nearly five derived fields over the entire dataset. In contrast, computing and materializing a scalar derived field for the entire dataset would require ∼5TB (15TB for vector fields).

Algorithm 1 Get points above threshold using cache

Require: Dataset d, Field f, Timestep t, Threshold k, Query box $q = [x_l, y_l, z_l, x_u, y_u, z_u]$

```
 1: procedure GetThreshold
 2:     points ← List()
 3:     updateCache ← false
 4:     query ← SELECT * FROM cachedb..cacheInfo
                WHERE dataset = d AND field = f
                AND timestep = t
 5:     command ← SqlCommand(query)
 6:     reader ← command.ExecuteReader()
 7:     if reader.HasRows() then
 8:         k_s ← reader["threshold"]        ▷ Stored threshold
 9:         start ← reader["startIndex"]
10:         end ← reader["endIndex"]
11:         ordinal ← reader["ordinal"]
12:         if k ≥ k_s & q ∈ [start, end] then
13:             query ← SELECT * FROM cachedb..cacheData
                    WHERE cacheInfoOrdinal = ordinal
14:             command ← SqlCommand(query)
15:             reader ← command.ExecuteReader()
16:             while reader.Read() do
17:                 location ← reader["zindex"]
18:                 norm ← reader["dataValue"]
19:                 if norm ≥ k & location ∈ q then
20:                     points.Add(new Point(location, norm))
21:                 end if
22:             end while
23:         else
24:             updateCache ← true
25:         end if
26:     else
27:         updateCache ← true
28:     end if
29:     if updateCache then
30:         Retrieve data covering q from DB.
31:         for all p ∈ q do
32:             Compute f at p.
33:             if ‖f(p)‖ ≥ k then
34:                 points.Add(new Point(p, ‖f(p)‖))
35:             end if
36:         end for
37:         Update cacheInfo and cacheData tables.
38:     end if
39:     return points
40: end procedure
```

The entire cache is comprised of two database tables. The *cacheInfo* table stores metadata for the cached entries. It stores information about the dataset, field, time-step, start and end coordinates of the spatial region examined and the threshold value used. The *cacheData* table stores the locations of all of the grid points, for which the field queried has a norm higher than the specified threshold. The *cacheData* table is foreign key constrained with the ordinal of the *cacheInfo* table. This allows us to quickly find a record in the *cacheInfo* table and retrieve all of the cached entries using an index lookup.

Overall execution of threshold queries: Algorithm 1 illustrates the process of obtaining all points with norms of the specified field above the given threshold from the database in the presence of a cache. The mediator submits a query to each of the database nodes storing the raw data asynchronously. Each node begins evaluation of the query by executing Algorithm 1. First a cache lookup is performed. If the data for the requested field, time-step and spatial region are available in the cache and if the specified threshold is higher than the one stored in the cache the query can be answered from there. The records are retrieved from the cache and the ones that have a higher value are returned to the mediator and subsequently back to the user. If the data stored in the cache have a higher threshold than the one requested the cache needs to be updated. Similarly, if the cache does not have an entry for the specified parameters the query needs to be evaluated from the raw data. In those cases the raw data are read into memory with data along the boundary requested from adjacent nodes as needed. The specified field is derived at each location on the grid and the norm or absolute value of the field is compared against the threshold. The locations where the values are higher than the threshold need to then be stored in the cache. If the cache does not have enough space for the new records, space is freed up by removing the least recently used data across all quantities. Reading from, updating or modifying the cache is done within a transaction with snapshot isolation level. Snapshot isolation allows us to avoid locking the tables that serve as the cache for each transaction. This provides for a higher degree of parallelism and avoids any potential deadlocks from queries running in parallel.

5. EXPERIMENTAL RESULTS

We evaluate the developed method for the execution of threshold queries to large numerical simulation datasets with the goal of analyzing the benefits and overhead from the introduction of the application-aware cache. We also analyze the scaling properties of the method. Finally, we show that an integrated method that performs the evaluation on the database nodes near the data is several orders of magnitude faster than the user requesting the derived filed of interest from the database and evaluating the threshold locally.

5.1 Experimental Setup

The experiments were run on the production database nodes of the JHTDB through a development Web-server hosting the Web-services. We used the MHD dataset (Sec. 2) for the experimental runs. This dataset is partitioned across 4 database nodes according to spatial regions in the Morton z-order. The database nodes are 2.66 GHz dual quad-core Windows 2008 servers with SQL Server 2008 R2 and 24 GB of memory. Each node has 24 2TB SATA disks arranged as a set of four RAID-5 arrays. The database files are striped across the nodes and their associated disk arrays. The tables storing the data are partitioned spatially along

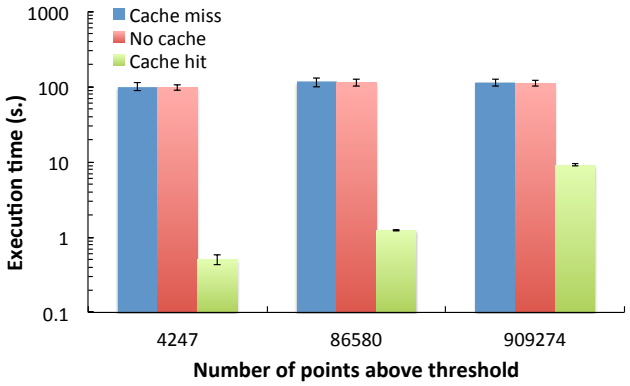


Figure 6: Execution time for threshold queries at different threshold levels compared with the execution time of the same queries in the absence of a cache.

contiguous ranges of the Morton z-curve and the data for each partition reside in one database file.

For this evaluation we looked at the performance of threshold queries to the vorticity field. The vorticity field is representative of derived fields that have to be computed from the stored data. It is defined as the curl of the velocity field. As described in section 3 thresholding the vorticity field is important in the study of fluid dynamics and obtaining the locations of maximum vorticity can lead to new insights into the development of the most intense vortices observed in the dataset.

5.2 Evaluation of cache effectiveness

The central part of the strategy that we have developed for the evaluation of threshold queries of derived fields is the application-aware cache, which stores the results of these queries. We first evaluate the overhead associated with the introduction and maintenance of the cache. Figure 6 compares the execution time of queries in the absence of a cache with the execution time of the same queries, which interrogate the cache first (blue and red bars in the figure). The execution times are also shown in Table 1. For these experiments we requested the locations with norms of the vorticity above thresholds at different levels. We refer the reader to Figure 2, which shows the distribution of values of the norm of the vorticity field in the MHD dataset to get an appreciation of the different threshold values used in the experiments. For the first set the threshold was set high (80.0) and only ∼4,300 points (or 0.0004% of all points) were above the threshold. For the second set a medium threshold (60.0) was chosen and ∼87,000 points (or 0.0081% of all points) were above the threshold. Finally, a low threshold (44.0) was chosen for the last set and there were ∼900,000 points (or 0.0847% of all points) above the threshold. For each set a random time-step was chosen and the queries were run against that time-step. The measurements were taken from the point of view of the end user.

As we can see from the results shown in Figure 6 the overhead associated with querying the cache first is minimal, less than 3% and within the margin of error. The cache was initially populated by executing several hundred unrelated queries and contained several million entries. During the "cache-miss" runs cache entries for the particular time-step queried were dropped before each run, making sure that each query would produce a cache miss and would have to be evaluated from the raw data. The execution times were averaged over 10 runs. We utilized 4 processes per database node for the evaluation of each query. The method shows stable running time across different time-steps and threshold levels in the absence of a cache and during cache misses. The running time increases slightly only because of the larger result set that has to be returned to the user.

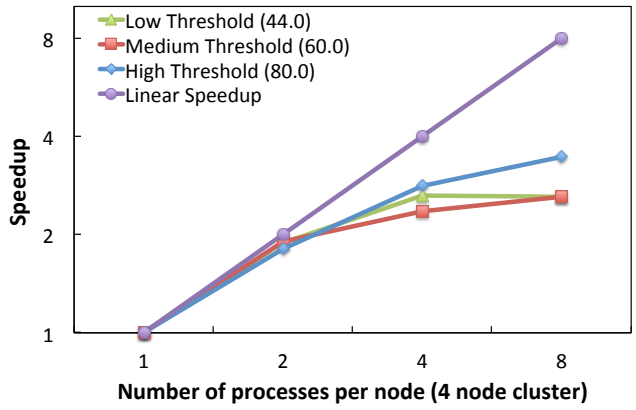
Vorticity threshold	Points above threshold	Average Running time (s.)		
		No cache	With cache (miss)	With cache (hit)
80.0	4247	97.1	100.2	0.5
60.0	86580	113.7	115.9	1.2
44.0	909274	111.6	115.0	9.1

Table 1: Effectiveness of caching.

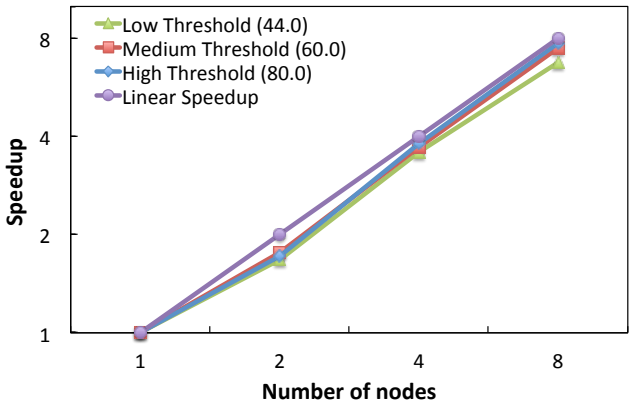
Cache hits reduce the running time of threshold queries by over an order of magnitude as shown in Figure 6 and Table 1. This is because we do not have to compute the requested derived field from the raw data, which eliminates the associated I/O. Only the cache entries need to be looked-up, which is substantially less data than the raw vector or scalar field data. For the queries with large result sets it is actually the network time taken to transfer the results to the user that dominates the overall execution as opposed to the I/O or computation time as we show later. Cache hits are evaluated by first warming up the cache by submitting the same set of threshold queries of the vorticity field as before. We then submit several more unrelated queries with different time-steps and threshold values in order to pollute the cache. Finally, we issue the original set of queries and measure their running times. Let us focus on the query with low threshold, which returns ∼900,000 points. Given that valid threshold values are limited to those that result in no more than 1,000,000 points it is likely that all subsequent queries to this time-step will result in a cache hit as their threshold is likely to be equal or higher than the cached one. Currently we observe fairly high cache-hit ratios as the workload is very structured and queries tend to examine the same regions in space and time.

5.3 Scaling and Distributed Evaluation

The evaluation of threshold queries of derived fields from the raw data is both I/O and computationally bound. These queries examine the entire data volume of a simulation time-step and are, therefore, good candidates for a data-parallel distributed evaluation. Our data-parallel implementation exhibits good vertical and nearly ideal horizontal scaling as shown in Figure 7. For the scale-up experiments (Fig. 7(a)), we used the same queries and threshold values as for the runs shown in Figure 6 and Table 1 but with varying number of processes per node. Cache entries for the time-step queried were again dropped before each run in order to evaluate the scaling properties of the computation of the derived field from the stored data. The computations for all of the derived fields of interest (such as the vorticity) at each grid point need data from adjacent grid points only. Therefore, each node of the cluster is able to compute the derived field



(a) Scale-up with multiple processes per node



(b) Scale-out to multiple nodes

Figure 7: Execution time for threshold queries at different threshold levels – high, medium and low. The scale-up evaluation was performed utilizing 1-8 processes per server on a 4-node cluster. The scale-out evaluation was performed on 1 through 8 nodes.

from data available locally with only a small amount along the boundary of each region having to be retrieved from adjacent nodes. Each computation is independent and embarrassingly parallel. This allows us to make use of multiple processes per node and scale out to multiple database nodes.

We observe nearly a two times speedup when going from a single process per node to two processes per node (Fig. 7(a)). The speedup diminishes to 1.4 times when going to 4 processes and little speedup is observed with 8 processes per node. While the computation time scales with increased process count, the time to perform I/O does not as the data on each node reside in the same database table and on the same set of disks. Additionally, I/O redundancy increases as the process count increases as data along the boundary of each region are requested by multiple processes. SQL Server already utilizes parallelism to perform the I/O even when data are retrieved utilizing a single query. Finally, the experiments were run on the live production database nodes, which were also servicing other user queries in addition to operating system and other SQL Server processes. Nevertheless, running with 4 processes per node is nearly 2.6 times faster when compared to running with a single process.

The scale-out experiments show a nearly perfect linear speedup as the evaluation is distributed to an increasing number of database nodes (Fig. 7(b)). For these experiments we issued queries with the same threshold levels as before to a cold cache. We utilized a single process per database node to evaluate the horizontal scaling of the computation. The evaluation benefits not only from the additional computational resources with the addition of database nodes to the cluster but also from the increased memory size. The data needed for the computation of each derived field are read into memory and the larger memory size means that there is less contention with other system and application processes and it is less likely that virtual memory needs to be used. SQL Server also benefits from a larger buffer pool, which reduces the I/O time.

As expected, we observe even weaker speedup when the queries perform nothing but I/O and the number of processes per node is increased. Figure 8 compares the running time of the queries with a medium threshold and executed

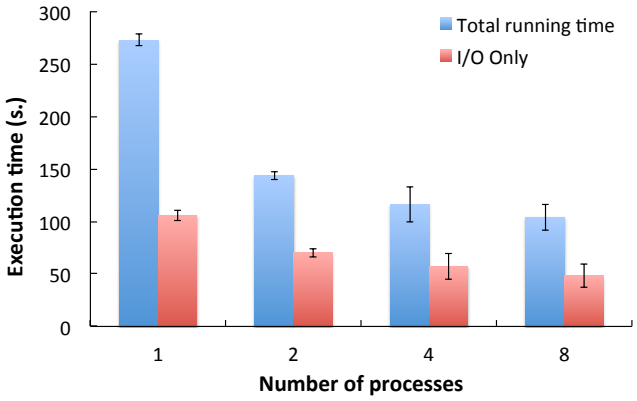
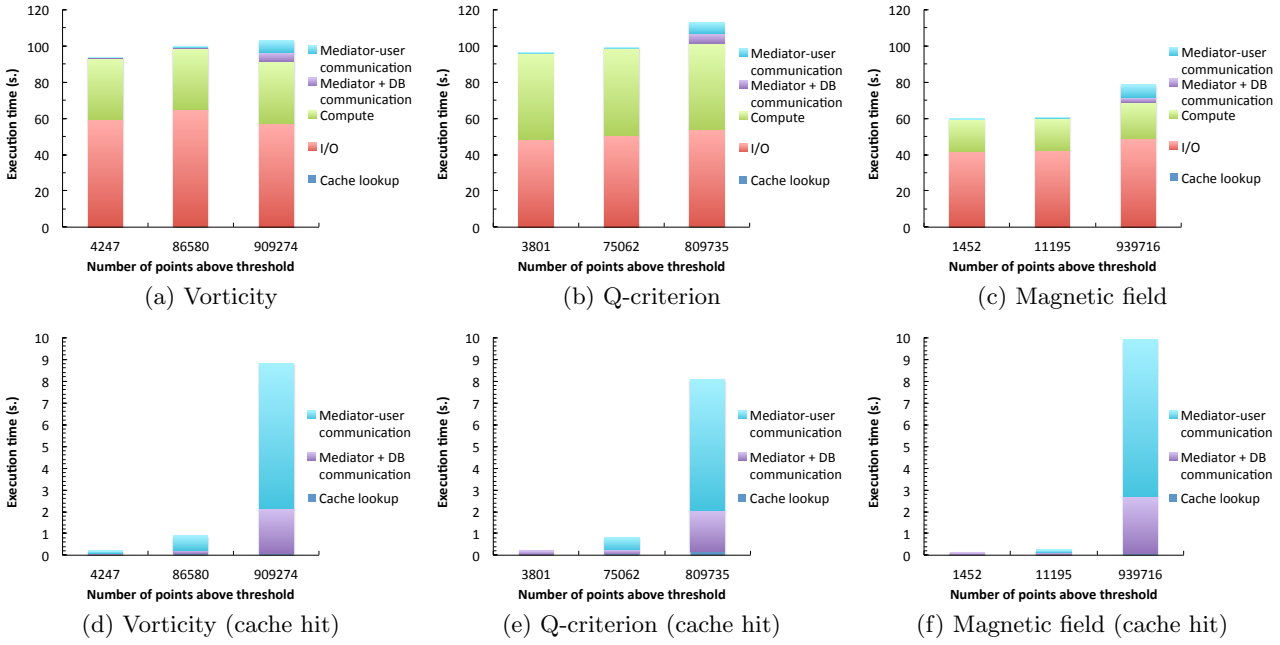


Figure 8: Execution time for threshold queries evaluated utilizing different number of processes per server compared with the time taken to perform the I/O only.

with varying number of processes per node with the time taken to perform the I/O only. The I/O time is about half of the total running time for these queries. SQL Server already makes use of parallelism internally and the data have to be retrieved from the same set of disks. Nevertheless, the I/O time does decrease with additional processes, this is because the data reside in a partitioned table and the data in each partition are placed in a separate file on one of the disk arrays. Depending on how the data requests are scheduled in SQL Server this allows for the disks arrays to be driven in parallel. Additionally, with more processes per node the data can also be consumed faster. It is worth noting that the total running time for the queries evaluated with 4 or 8 processes is about the same as the time it takes to perform the I/O only with a single process.

So far we have presented the effectiveness of evaluating threshold queries of derived fields on the database cluster storing the raw simulation data. The data-parallel computation of the derived fields allows us to evaluate a threshold



(a) Vorticity (b) Q-criterion (c) Magnetic field

(d) Vorticity (cache hit) (e) Q-criterion (cache hit) (f) Magnetic field (cache hit)

Figure 9: Breakdown of the execution time for threshold queries requesting different fields and at different threshold levels – high, medium and low.

query over an entire 1024^3 time-step part of a 20TB dataset in less than two minutes. The introduction of an application-aware cache for the query results of these queries reduces this time to several seconds when there is a cache hit. In contrast, one of our science collaborators reported that his evaluation of this functionality performed locally would take over 20 hours to complete. To perform the evaluation locally the user requests the derived field of interest from the database by submitting multiple queries over subregions of a time-step. This is necessary as requesting a derived field over an entire time-step will overload the network. Derived fields may have even more components than the scalar or vector field stored in the database. For example, the velocity gradient (needed for the computation of the vorticity) has 9 components compared with the 3 components of the velocity. Given a single-precision floating-point representation, this makes the velocity gradient of an entire time-step at least 36GB in size. A Web-service request will be much larger due to the overhead of wrapping the data in an xml format. After the field of interest is obtained locally the user has to threshold it to get the final result, which is reasonably fast, but discards most of the data that have been requested to yield a small in size result.

5.4 Evaluation of Additional Fields

The data-parallel evaluation of threshold queries shows stable execution time for different derived fields in addition to the different threshold levels and time-steps queried. The execution times depend on the complexity of the computation needed to evaluate the particular derived field requested. Figures 9(a), 9(b) and 9(c) show a breakdown of the execution time of threshold queries of different derived fields, which are evaluated from the raw data and a cold cache using 4 processes per node on a 4 node cluster. Almost the entire time is spent performing the I/O and computation associated with the derived field requested.

The vorticity field and the Q-criterion have similar I/O requirements as they have the same kernels of computation and are both derived from the velocity gradient. The vorticity has 3 components and its computation only examines 6 of the 9 components of the velocity gradient, which are also examined in pairs (see Eq. 1). On the other hand, even though the Q-criterion is a scalar value, it is computed through a non-linear combination of all 9 of the components of the velocity gradient. This means that the velocity gradient has to be computed at each grid location before the Q-criterion is evaluated, which is reflected in the increased computation time that we observe for the Q-criterion. The magnetic field is one of the raw fields of the magnetohydrodynamics dataset that are stored in the database. Therefore, there is no additional computation needed to derive it from the data, every data point has to be simply compared with the threshold level specified. This is why the computation time is much smaller compared to the queries for the vorticity and the Q-criterion. The I/O time for the magnetic field is also smaller. This is because its kernel of computation is a single point and therefore there are no additional data along the boundary that have to be requested from adjacent nodes. In that case all of the data needed are available locally for each database node.

In all of these cases, the time taken to interrogate the cache is negligible. The mediator time to distribute the queries and assemble the results as well as the time to transfer them to the user are also substantially smaller than the I/O and computation times. As expected they increase proportionally to the number of points in the result set.

It is interesting to note that the time taken to perform a cache lookup is relatively small even in the case of a cache hit as can be seen in Figures 9(d), 9(e) and 9(f). This is because the cache tables reside on SSDs attached to each

database node (see Fig. 5) and retrieving the data is always done through a clustered index lookup. In the cases of a cache hit, the majority of the time is spent simply transferring the results from the database nodes to the mediator and then back to the user. These times remain more or less constant between the cases of cache misses and cache hits (top row and bottom row of Fig. 9). Caching the results of threshold queries effectively preserves the I/O and computational effort spent during their initial evaluation and results in over an order of magnitude speedup for all the different fields requested as we can see in Figure 9.

6. RELATED WORK

Only select few database systems offer support for arrays as first-class citizens. Even fewer provide the fault-tolerance, scalability and availability guarantees necessary for a system managing multi-terabyte datasets in a production setting. This is part of the reason why we have chosen to represent the array data in the JHTDB as a collection of binary large objects in a relational DBMS and perform the array manipulation tasks necessary at the application level. The systems that provide support for arrays and aim to handle large array data efficiently are RasDaMan [5], SciDB [30] and MonetDB/SciQL [34]. RasDaMan partitions raster objects into tiles, which are stored in a traditional relational database system. This approach is similar to how the numerical simulation data are handled in the JHTDB. RasDaMan provides RasQL [4], which is a SQL-92 based query language for the manipulation of raster images. SciDB is an array database system build from the ground up. Array attributes are partitioned vertically and each attribute array is decomposed into overlapping chunks. SciDB provides a declarative Array Query Language (AQL) and an Array Functional Language (AFL). Users can create arrays with named dimensions with AQL and make use of the functional operators defined in AFL, such as SLICE, SUBSAMPLE, SJOIN, FILTER and APPLY. SciQL's focus is on language design and integration with SQL:2003 syntax and semantics. It is implemented within the MonetDB framework [24].

Database systems support rollup queries, including top-k queries, but in most cases these queries apply only simple linear score functions on the attribute values of individual records. Additionally, many top-k query evaluation techniques rely on the score functions being monotone in order to perform early pruning (see [15] for a survey of top-k evaluation strategies). This is an assumption that we cannot make for the functions used to compute all the different possible derived fields of interest in fluid mechanics. Even approaches that aim to work with general score functions [11, 33] assume that the function operates on the attributes of a single record. In contrast, our approach performs a kernel computation at each grid location in order to obtain the value of a *derived* field at that location and examines the vector or scalar array data at all neighboring locations, which are within the kernel of the computation. The functions used to derive the field may even be non-linear. Finally, a top-k approach may not be suitable in the cases where scientists are interested in performing threshold queries at different time-steps as the same threshold level will produce different number of points in the result set for different time-steps.

The processing of top-k queries has been studied extensively in the context of distributed and relational database systems. A survey of different techniques in the case of centralized processing is given in [15]. In the case of distributed processing different approaches focus on horizontally [3, 32] or vertically [7, 8, 13, 21, 22] distributed data. None of these approaches deal with array data stored in a relational database system. Zhao et al. propose an algorithm for the processing of top-k queries in large-scale distributed environments called BRANCA [35]. They build on the idea of semantic caching [27] and make use of branch caches, which store results of previous top-k queries with respect to the data stored on each server. The caching mechanism that we use is similar in that regard, but the queries that are evaluated in our system operate on derived fields, which are computed at each location by accessing data from a surrounding region. The queries described in [35] operate over the attributes of individual records only using simple linear score functions.

Aßfalg et al. introduce the concept of threshold queries in time-series databases [2]. Their definition of threshold queries differs from the threshold queries described in this paper. They are concerned with determining the time-series, which exceed a user-defined threshold at time frames similar to the time-series specified in the query. Thus, their definition of threshold queries is concerned with the temporal relationship between the time-series stored in the database (usually one dimensional sequence of measurements) and the time-series given in the query. In contrast, our approach focuses on reporting all of the spatial locations of a multidimensional field where the norm or absolute value of the field exceeds a user prescribed threshold.

In a system called the tree cache, Lopez et al. [20] make use of a small application-aware cache to reduce access time to large datasets stored on disk. The tree cache stores individual octants of octree datasets and exploits application-specific information to determine which octants to cache and to perform query reordering. This work has inspired the use of an application-aware cache for the evaluation of threshold queries. In contrast to the tree cache, we do not cache raw data objects, but rather query results. Caching query results preserves the computational effort in addition to reducing I/O, which has a much bigger impact on query performance and substantially reduces the size of the cache. Additionally, the cache that we introduce resides on disk rather than in memory, which greatly increases its potential size. Lopez et al. also explore approximate querying through aggregation, which can be fairly easily supported by our system but is of limited use as scientists performing threshold queries are usually interested in obtaining the exact locations where a field is at its highest values.

Sampling approaches [29, 25] offer an alternative to the on-demand computation of derived fields and the evaluation of threshold queries on them. The goal of both techniques is to not return large data volumes, but focus on the most intense events and interesting regions in the dataset. The computation of derived fields is carried out on the nodes of the database cluster and takes a look at the dataset as a whole, while the user obtains only a small subset of the data, where the derived field in question is above the prescribed threshold. Sampling approaches can potentially omit some locations and while useful for generating initial impressions may not be suitable if the exact locations where a field is at its highest values are desired.

Andrade et al. [1] describe a database system and an optimization framework build on the concept of *active se-*

mantic caching. An active semantic cache aims to fully or partially reuse cached query results or aggregates through automated transformations of these aggregates. Similarly to our work they focus on real scientific data-analysis applications. The method that we have developed for the evaluation of threshold queries complements the active semantic caching approach and could be used in that framework. Our work has focused on extending a relational database system (Microsoft's SQL Server) as opposed to designing a new database system from the ground up as described by Andrade et al. [1].

7. CONCLUSIONS AND FUTURE WORK

We have presented an efficient strategy for the evaluation of threshold queries of derived fields in large numerical simulation datasets. The thresholded fields are derived from the stored simulation data in a distributed data-parallel manner. The computations scale with the cluster resources and are performed on the database nodes, where the data are stored. This new capability allows researches to quickly obtain and focus on regions of special interest even if they lack the computing capabilities or data transfer rates necessary to examine entire time-steps or large parts of the entire dataset.

We have introduced an application-aware cache for the query results of threshold queries. Cache hits reduce query running times by over an order of magnitude. The cache adds minimal overhead during the evaluation of queries even if there is a cache miss and has modest storage requirements. The cache is represented as a set of database tables and resides on disk rather than in memory. Each database node has local cache tables, which allows the cache to scale-out as the cluster grows.

The introduction of an application-aware cache for query results lays the groundwork for the creation of a landmark database. Such a database can store the locations of the highest vorticity regions in the dataset or more broadly regions of interest and their associated statistics.

The Web-services approach to archived numerical simulation datasets provides public access to high quality simulation data to anyone with an internet connection. The Web-services methods can be called from any modern programming language and we provide C, Fortran and Matlab client libraries for the JHTDB. The evaluation of each query submitted through a Web-service call is carried out on the nodes of the database cluster by means of a stored procedure or a user-defined function that has been implemented and deployed to handle these requests. This allows us to fine-tune the execution of these procedures and handle all requests transparently to the user. However, this approach also has drawbacks. Adding new functionality means adding to a long list of Web-service calls and requires substantial implementation effort. In the case of threshold queries the stored procedure performing the evaluation must have an implementation for each derived field of interest even though the execution is handled by the same Web-service call.

In the future, we plan to develop declarative and graphical user interfaces that will allow users to combine existing building blocks and perform computations that have not been explicitly implemented. Additionally, we plan on deploying a server-side computing environment for users similar to the CasJobs service for the Sloan Digital Sky Survery [18]. In such an environment users can run queries in batch mode and save their results in a personal database called MyDB, which resides on the servers near the data. This will allow for much greater flexibility in the type of computations that can be performed in addition to substantially decreasing the network overhead.

8. ACKNOWLEDGMENTS

The authors would like to thank the Turbulence Database Group at Johns Hopkins University as well as three anonymous reviewers for their insightful comments and suggestions. This work is supported in part by the National Science Foundation under Grants CMMI-0941530, ACI-1261715, OCI-1244820 and AST-0939767 and Johns Hopkins University's Institute for Data Intensive Engineering & Science.

9. REFERENCES

[1] H. Andrade, T. Kurc, A. Sussman, and J. Saltz. Active semantic caching to optimize multidimensional data analysis in parallel and distributed environments. *Parallel Computing*, 33(7-8):497–520, Aug. 2007.

[2] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity Search on Time Series based on Threshold Queries. In *EDBT*, 2006.

[3] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *ICDE*, 2005.

[4] P. Baumann. A Database Array Algebra for Spatio-Temporal Data and Beyond. In *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems*, NGIT '99, 1999.

[5] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The Multidimensional Database System RasDaMan. In *SIGMOD*, 1998.

[6] R. Burns, K. Lillaney, D. R. Berger, L. Grosenick, K. Deisseroth, R. C. Reid, W. G. Roncal, P. Manavalan, D. D. Bock, N. Kasthuri, M. Kazhdan, S. J. Smith, D. Kleissas, E. Perlman, K. Chung, N. C. Weiler, J. Lichtman, A. S. Szalay, J. T. Vogelstein, and R. J. Vogelstein. The open connectome project data cluster: Scalable analysis and vision for high-throughput neuroscience. In *SSDBM*, 2013.

[7] P. Cao and Z. Wang. Efficient top-K Query Calculation in Distributed Networks. In *PODC*, 2004.

[8] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing Top-k Selection Queries over Multimedia Repositories. *IEEE Trans. on Knowl. and Data Eng.*, 16(8):992–1009, Aug. 2004.

[9] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic Data Caching and Replacement. In *VLDB*, 1996.

[10] DataScope. http://idies.jhu.edu/datascope.

[11] P. M. Deshpande, D. P, and K. Kummamuru. Efficient Online top-K Retrieval with Arbitrary Similarity Measures. In *EDBT*, 2008.

[12] G. Eyink, E. Vishniac, C. Lalescu, H. Aluie, K. Kanov, K. Bürger, R. Burns, C. Meneveau, and A. Szalay. Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. *Nature*, 497(7450):466–9, 2013.

[13] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB*, 2000.

[14] A. J. G. Hey, S. Tansley, and K. M. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[15] I. F. Ilyas, G. Beskales, and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, Oct. 2008.

[16] K. Kanov, R. Burns, G. Eyink, C. Meneveau, and A. Szalay. Data-intensive Spatial Filtering in Large Numerical Simulation Datasets. In *Supercomputing*, 2012.

[17] K. Kanov, E. Perlman, R. Burns, Y. Ahmad, and A. Szalay. I/O Streaming Evaluation of Batch Queries for Data-intensive Computational Turbulence. In *Supercomputing*, 2011.

[18] N. Li and A. R. Thakar. CasJobs and MyDB: A Batch Query Workbench. *Computing in Science and Engineering*, 10(1):18–29, 2008.

[19] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, page N31, 2008.

[20] J. C. Lopez, D. R. O'Hallaron, and T. Tu. Big Wins With Small Application-aware Caches . In *Supercomputing*, 2004.

[21] A. Marian, N. Bruno, and L. Gravano. Evaluating Top-k Queries over Web-accessible Databases. *ACM Trans. Database Syst.*, 29(2):319–362, June 2004.

[22] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A Framework for Distributed Top-k Query Algorithms. In *VLDB*, 2005.

[23] The Millennium Simulation. http://www.mpa-garching.mpg.de/millennium/.

[24] MonetDB. http://monetdb.cwi.nl/.

[25] S. Nirkhiwale, A. Dobra, and C. Jermaine. A sampling algebra for aggregate estimation. *Proc. VLDB Endow.*, 6(14):1798–1809, Sept. 2013.

[26] E. Perlman, R. Burns, Y. Li, and C. Meneveau. Data Exploration of Turbulence Simulations Using a Database Cluster. In *Supercomputing*, 2007.

[27] Q. Ren, M. H. Dunham, and V. Kumar. Semantic Caching and Query Processing. *IEEE Trans. on Knowl. and Data Eng.*, 15(1):192–210, Jan. 2003.

[28] The Sloan Digital Sky Survey. http://www.sdss.org/.

[29] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, 2011.

[30] M. Stonebraker, J. Becla, D. J. DeWitt, K. Lim, D. Maier, O. Ratzesberger, and S. B. Zdonik. Requirements for Science Data Bases and SciDB. In *CIDR*, 2009.

[31] A. S. Szalay, G. Bell, J. Vandenberg, A. Wonders, R. Burns, D. Fay, J. Heasley, T. Hey, M. Nieto-Santisteban, A. Thakar, C. van Ingen, and R. Wilton. GrayWulf: Scalable Clustered Architecture for Data Intensive Computing. In *HICSS*, 2009.

[32] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. On Efficient Top-k Query Processing in Highly Distributed Environments. In *SIGMOD*, 2008.

[33] D. Xin, J. Han, and K. C. Chang. Progressive and Selective Merge: Computing Top-k with Ad-hoc Ranking Functions. In *SIGMOD*, 2007.

[34] Y. Zhang, M. Kersten, M. Ivanova, and N. Nes. SciQL: Bridging the Gap Between Science and Relational DBMS. In *IDEAS*, 2011.

[35] K. Zhao, Y. Tao, and S. Zhou. Efficient Top-k Processing in Large-scaled Distributed Environments. *Data Knowl. Eng.*, 63(2):315–335, Nov. 2007.