Exploring Cloud Opportunities from an Array Database Perspective

Alex Dumitru
Jacobs University Bremen
College Ring 1
28759 Bremen, Germany
+494212003139
m.dumitru@jacobs-university.de

Vlad Merticariu
Jacobs University Bremen
College Ring 1
28759 Bremen, Germany
+494212003139
v.merticariu@jacobs-university.de

Peter Baumann
Jacobs University Bremen
College Ring 1
28759 Bremen, Germany
+494212003178
p.baumann@jacobs-university.de

ABSTRACT
Since array data of arbitrary dimensionality appears in massive amounts in a wide range of application domains, such as geographic information systems, climate simulations, and medical imaging, it has become crucial to build scalable systems for complex query answering in real time. Cloud architectures can be expected to significantly speed up array databases.

We present an enhancement of the well-established Array DBMS rasdaman with intra-query distribution capabilities: requests, incoming in the form of database queries, are broken into sub-queries which are then distributed in a network of known peers. The splitting strategies ensure that the network's processing power is fully exploited, while at the same time enabling optimizations such as network traffic minimization.

Categories and Subject Descriptors
C.2.4 [**Distributed Systems**]: Client/server
E.1 [**Data Structures**]: Arrays
H.2 [**Database Management**]: Query languages

General Terms
Algorithms, Performance, Design, Reliability, Experimentation.

Keywords
Distributed array databases, rasdaman, Big Data.

1. INTRODUCTION
Multi-dimensional, high-volume arrays appear in a large variety in manifold domains, such as medical imaging, geographic information system, environmental and astronomical observations, or high precision simulations of physical phenomena [6][16]. While in classical relational systems tuples are well below database page size, single array objects easily exceed today's server RAM, such as 4-D climate simulation data cubes with dozens of Terabytes. Operations on arrays are numerically quite expensive, and consequently array database queries normally are CPU-bound as opposed to the often IO-bound relational query evaluation. Therefore, efficient processing is a critical factor for the overall query response time.

Beyond query optimization [15], parallel query processing is the most promising technique to speed up complex operations on large data volumes [5]. Intra-query parallelism is a well-established mechanism for achieving high performance in relational database systems. However, specific properties of arrays require different approaches in the attempt of applying the same methods on array database systems. The most characterizing property of arrays is the well-defined Euclidean neighborhood of cells, which has high impact on access locality (when a particular cell is accessed it is extremely likely that its neighbor pixels will get accessed, too) and induces efficient partitioning techniques for storage, like tiling and [11] chunking [12].

In this contribution we describe the extension to an established Array DBMS, rasdaman [13], that takes advantage of the shared-nothing cloud architecture in order to distribute the processing resulting from complex, array based operations. The preliminary runs show promising results in terms of processing speed-up and scalability.

The remainder of this paper is organized as follows. In Section 2 we review the state of the art. Our approach to distributed array query processing in a cloud, the rasdaman federation, is introduced in Section 3. In Section 4, first performance evaluation results are presented, and Section 5 gives a conclusion.

2. STATE OF THE ART
Exploiting parallelism to process queries has been widely explored for improving query response times [1][3]. Different strategies and algorithms have been devised to address this issue in relational databases. As RDBMSs are usually IO-bound, most of them focus either on partitioning the data and applying the same query execution tree to smaller partitions or distributing a limited set of operators to different processors on the same machine [2].

Array support for databases has only recently found major interest in the database community, although relevance has been claimed already early by Maier [17], and the fully multi-dimensional Array Algebra [7] and its implementation, rasdaman [13], have been around for some time. Meantime, several more array database approaches exist, and a first workshop dedicated to this class of DBMSs has been held [14].

ArrayStore is a storage manager designed for large multi-dimensional arrays [4]. It supports parallel processing of an array on several nodes simultaneously. However the intra-query

DanaC'14, June 22 2014, Snowbird, UT, USA

ACM 978-1-4503-2997-2/14/06…$15.00.
http://dx.doi.org/10.1145/2627770.2627775

functionality is provided by applying the entire set of operations on each pre-partitioned chunk and merging the results.

An earlier attempt at implementing intra-query parallelization in rasdaman was done using MPI for inter-process communication [5]. It defined two types of servers, *Scheduling Servers* that receive queries and distribute them across the network and *Slave Servers* used for answering query fragments. However, this implementation restricts the scalability of the system as the number of servers from each type has to be carefully chosen based on the load in the system and the number and type of queries being executed. Further, it has a single point of failure.

The approach reported here uses a different paradigm, a shared-nothing architecture in which queries are dynamically split into execution units – i.e., subqueries – which are shipped to the data sites. Queries are split in a way that minimizes intermediate result transfer between the participating nodes.

3. THE RASDAMAN FEDERATION

3.1 The rasdaman engine

In this section we offer a brief overview of the rasdaman Array DBMS, as far as required for this discussion. The complete rasdaman query language, algebra, architecture, and impact on standardization are presented elsewhere [7][8][9].

In rasdaman, arrays are modeled as function mappings form an n-dimensional domain to a value set. Arrays are typed – with atomic or structured types – and have an extent that can have fixed or variable bounds in each direction.

Anticipating its embedding into the relational model, arrays in rasdaman are grouped into tables (called collections) with two columns, one holding the array and the other one an object identifier. In this respect, rasdaman can be seen as a column store specialized towards arrays.

The rasdaman query language [7], *rasql*, respects the set-oriented SELECT/FROM/WHERE paradigm of SQL and extends it with n-D array operators. These are based on the rasdaman Array Algebra, a minimal, safe, and optimizable framework for array processing.

As an example, the following query iterates over all arrays in collection *ExampleImageSet*, filters those where the average nir (near-infrared) value exceeds threshold 200, and returns a TIFF of the band ratio between the red and nir channel for a given bounding box:

```
SELECT encode(
       ((m.red - m.nir) / (m.red + m.nir ))
       [0:1000,0:1000],
       "image/tiff" )
FROM   ExampleImageSet AS m
WHERE  avg_cells( m.nir <= 200)
```

Prior to evaluation, queries undergo heuristic optimization based on about 150 rewriting rules [15]. Finally, queries are processed using the "tile streaming" paradigm so that large objects are loaded and evaluated only piecewise.

3.2 Concept description

Among the parallelization techniques that rasdaman implements, the latest one is intra-query parallelization where incoming queries are transparently split and distributed over a network of known peers. In the effort of adapting the engine to cloud processing patterns, two main components have been identified: one handling communication and gathering information about the collections sitting on every server, and a processing unit responsible with dividing work among available nodes and executing operations on local data. Each node has an instance of both.

The federation daemon collects and stores statistics from the other network nodes and provides real time updates about local changes. Information exchanged includes available datasets, CPU load, and memory usage per rasdaman host. In a consistent state, all federation daemons in a system will store the same information.

The processing component is represented by the rasdaman server. Using information from the federation daemon it breaks queries into sub-queries to be executed on peer nodes, ships them, and assembles the intermediate results provided by the nodes into the final query result. Actually, the rasdaman server structure is more involved, but we omit the details as they do not contribute to the discussion on hand.

Thus, we define a rasdaman network node as a pair of a federation daemon and a rasdaman server. A rasdaman peer network, then, is a collection of one or more nodes that can communicate with each other. Any peer can receive a query and will subsequently act as this query's dispatcher, so all peers are at the same level and there is no single point of failure. Should a node become inaccessible then the peers will recognize this and will not any longer consider this peer for distribution. Conversely, a peer at any time can join the network.

An important aspect of the rasdaman network is its flexibility. In the next subsections we will describe each component in detail and explain how network settings can be tweaked to match different scenarios.

3.3 Communication Model

As mentioned above, communication between nodes is maintained by the federation daemon. Its main purpose is to ensure consistency between the nodes in the network by keeping track of changes in the local metadata and propagating them timely throughout the network. Three design goals have received particular attention:

3.3.1 Robustness

The federation daeemon needs to be highly resilient against internal (wrong configuration parameters, implementation mistakes) and external (network delay, dependent services failing) errors. Failures are considered regular events in the workflow that can happen at any time. Internally the daemon is divided into subservices, each performing a specific task such as metadata tracking, statistics gathering, etc. Each node is independent and is automatically restarted upon failures. This allows the service to function well in face of occasional localized failures which are expected in cloud environments, thereby making it highly robust. Furthermore, the rasdaman engine is designed to continue serving local collections even in case the daemon stops working completely for some reason.

3.3.2 Consistency

Consistency of the metadata in the network is another important quality. As the queries that rasdaman executes usually involve long running processes dealing with large sets of data being executed on multiple servers, incorrect metadata can lead to queries failing or returning wrong results.

The nodes send status messages regularly, exchanging information about the metadata that is stored on the local node plus their knowledge of the overall network. Each node keeps in its own registry a list of the local metadata objects together with a sequence number incremented each time a change is detected. A status message consisting of the local status along with the information received from the other nodes in the network is broadcast at regular intervals. When a node receives a status message it decides if it

needs to update its registry entry for the sender node based on the following algorithm:

- If the sequence number received is larger than the one kept in the registry then the local entry is updated.
- If the sequence number is identical to the local one but the object is different, a simple majority vote in the network is started to decide which metadata information is correct.
- If sequence number and object are identical to the local registry information, no action is taken.

Furthermore, at each metadata change, a status message is automatically broadcast to ensure prompt consistency.

3.3.3 Flexibility

One of the main qualities of the rasdaman engine is the flexibility that it gives system administrators to adjust the system in order to increase performance and adapt to the current environment, for example by choosing the storage method or the tiling scheme.

The federation daemon continues this tradition by offering administrators both the option of configuring the communication parameters (status message frequency, node timeout interval etc) and the network topology. Nodes can be grouped into two categories, which can be combined:

- Inpeer – the current node accepts requests from the node listed as an inpeer but abstains from sending requests to it.
- Outpeer – the current node can send requests to the outpeer node but is not allowed to receive requests from it.

This can be exploited by administrators to create complex network topologies both inside internal cloud networks and between external networks. Data centers can use this feature to enable or disable communication patterns between them based on data access agreements. It also allows them to scale their connection by adding or removing entry points, on-demand.

3.4 Query Distribution Model

A rasdaman server receiving a query splits and distributes it in two steps: first the query is broken down, and then the optimal server is chosen for the execution of each subquery.

3.4.1 Query breakdown

Goal is to divide the query into sub-queries that can be executed independently on different federation nodes (again, we ignore parallelization inside a node in this context). The following algorithm maximizes parallelization:

1. For each object addressed in the query received, inquire the federation daemon to determine set of servers on which the object is available. Each object access node in the query tree is tagged with the set of nodes offering it.
2. Starting from each object access node the query tree is walked upwards to combine larger processing units. The node set tags are propagated accordingly. When tree branches meet, their tag sets are intersected to yield the parent node's tag set.
3. When a tag set gets empty then the distribution algorithm stops, and considers the children as independent subqueries to be distributed to some node listed in the tag set.

The algorithm ensures that the sub-trees to be executed remotely are maximal. This basic algorithm can be modified for different goals. For example, we consider the data flow along each operator tree edge and determine breakpoints with minimal transport costs. This way, network traffic can be optimized.

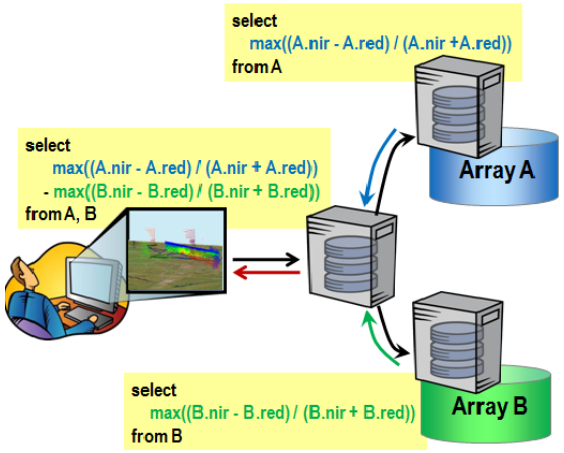

Figure1. Sample distribution of a rasdaman query in a peer network.

Fig. 1 shows a scenario where the user wants to derive the difference between the NDVI maxima of images A and B; NDVI (Normalized Difference Vegetation Index) belongs to the family of band ratio algorithms and allows, by comparing the nir (near-infrared) and red bands of a satellite image, to determine the probability of a pixel containing vegetation. The receiving node doesn't host neither A or B, however it knows, through the federation daemon, where they sit. Therefore, sub-requests can be spawned to fetch information from these servers. The split doesn't take place at the bottom of the tree (i.e. where data are loaded into the engine), but a position in the query tree where the data traffic is minimized. In the example on hand, the `max` aggregate delivers a single number, hence is an optimal point for sub-query generation. The nodes A and B compute the maximum NDVI over each array in parallel and send back the resulting two values. The receiving node performs the subtraction and sends the result back to the user.

3.4.2 Determining the optimal server

After identifying the query sub-trees that can be executed remotely, each one is labeled with a set of possible network destinations. This set is determined in the query breakdown phase and contains the servers that have all the objects affected by the sub-tree available.

When more than one server is available for the execution of a sub-tree, the best one is chosen based on the statistics offered by the federation daemon. Preferences can be set by the network administrator, depending on the anticipated query load. As queries usually are CPU-bound a common strategy is to select the server with the most CPU resources available [16]).

4. PRELIMINARY RESULTS

While rigorous performance evaluation is under work, we already have made first measurements to assess feasibility of our approach. To this end, a rasdaman federation has been deployed in the Amazon Elastic Cloud (EC2) environment [10]. Test queries have been executed in a series of scenarios varying in the number of network nodes. Each node instance was of type "micro"[1]. The test

[1] single core 2007 Xeon processor 1.2 GHz, 615 MB RAM

data set consists of a 2-D RGB image of 9 million pixels representing the visible channels of a hyperspectral satellite image.

We have measured the execution time of two queries:

- "the average over all the image cells":

```
SELECT avg_cells(a) FROM satImages a
```

- "the histogram over the red channel":

```
SELECT MARRAY x in [0:256]
       VALUES count_cells(a.red = x)
FROM   SatImages as a
```

Fig. 2 shows the results obtained. We observe good speedups for both queries. However, the average query computing is clearly affected by the overhead induced by the query breakdown and data shipping as the number of servers grows. The second query is much more computationally intensive, hence the overhead gets negligible. This indicates several dimensions that need to be taken into account when estimating the system's scalability, some of which we will discuss in the conclusion below.

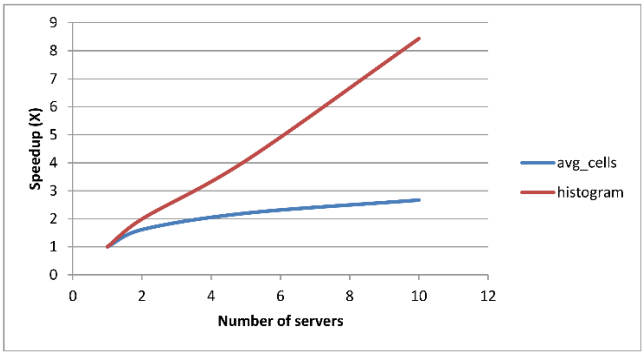

Figure 2. Query execution time for different network sizes.

5. Conclusion

We have presented a system which, given a SQL style query on array data of any dimensionality, breaks it into sub-queries distributed to a network of known peers, taking full advantage of the available processing power. Several optimizations have already been implemented (such as minimizing the network traffic or sending requests to the servers having the most available CPU), however, as described below, there are still many to be applied.

In this paper, we tried to determine whether cloud computing may be used to speed up array database queries. Since the execution time is usually heavily influenced by the CPU power, we believe that the answer is yes: cloud computing represents a great opportunity for speeding-up Array DBMS queries.

Future work plans include two main topics: transparent query object distribution. Currently, in order to distribute an object over several servers, the user has to split it manually and insert each part locally.

Both these features can be automated. As a first step, we plan to extend rasdaman's query rewriting engine with rules targeting distributed objects. Then we will investigate different partitioning strategies, which we hope will stay at the basis of our automated data ingestion process.

ACKNOWLEDGMENTS

Part of this work has been supported through EU FP7 EarthServer (www.earthserver.eu). Thanks to George Merticariu for the cloud benchmarking suite.

6. REFERENCES

[1] Hamid Pirahesh, C. Mohan, Josephine Cheng, T. S. Liu, and Pat Selinger. Parallelism in relational data base systems: architectural issues and design approaches DPDS, ACM, New York, NY, USA, 1990.

[2] De Witt, D.J., " A Multiprocessor Organization for Supporting Relational Database Management Systems," Computers, IEEE Transactions on , vol.C-28, no.6, pp.395,406, June 1979.

[3] Hong, Michael A. Olson Wei Michael, and Michael Ubell Michael Stonebraker. "Query processing in a parallel object-relational database system." Data Engineering: 3, 1996.

[4] Soroush, Emad, Magdalena Balazinska, and Daniel Wang. "Arraystore: a storage manager for complex parallel array processing." Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011.

[5] Hahn, Karl, and Bernd Reiner. "Intra-query parallelism for multidimensional array data." 28th International conference on very large data bases (VLDB), Hongkong. Vol. 20. 2002.

[6] Baumann, Peter, et al. Geo/environmental and medical data management in the RasDaMan system. VLDB. 97, 1997.

[7] Peter Baumann: A Database Array Algebra for Spatio-Temporal Data and Beyond. The Fourth International Workshop on Next Generation Information Technologies and Systems (NGITS '99), Zikhron Yaakov, Israel, Lecture Notes on Computer Science 1649, Springer Verlag, July 5-7, 1999.

[8] Baumann, Peter. "The OGC web coverage processing service (WCPS) standard." Geoinformatica 14.4 (2010): 447-479.

[9] Baumann, Peter, et al. "The multidimensional database system RasDaMan." ACM SIGMOD Record, Vol. 27, No. 2. ACM, 1998.

[10] Amazon Web Services. Amazon Elasic Compute Cloud - User Guide. API Version. 15.10.2013.

[11] P. Baumann, S. Feyzabadi, C. Jucovschi: Putting Pixels in Place: A Storage Layout Language for Scientific Data. Proc. IEEE ICDM Workshop on Spatial and Spatiotemporal Data Mining (SSTDM-10), pp. 194 – 201, December 14, 2010.

[12] S. Sarawagi, M. Stonebraker. Efficient Organization of Large Multidimensional Arrays. Proc. ICDE, Washington, DC, USA, pp. 328-336, 1994.

[13] Peter Baumann: On the Management of Multidimensional Discrete Data. VLDB Journal 4, Special Issue on Spatial Database Systems, pp. 401-444, 1994.

[14] P. Baumann, B. Howe, K. Orsborn, S. Stefanova: Proceedings of the 2011 EDBT/ICDT Workshop on Array Databases, Uppsala, Sweden, March 25, 2011 ACM 2011.

[15] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, Norbert Widmann: Spatio-Temporal Retrieval with RasDaMan. Very Large Data Bases(VLDB), Sep 7-10, 1999.

[16] Rusu, Florin, and Yu Cheng. "A Survey on Array Storage, Query Languages, and Systems." arXiv preprint arXiv:1302.0103, 2013.

[17] Maier, David, and Bennet Vance. "A call to order." Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM, 1993.