A Benchmark for Interactive Data Cubes

Sean Stephens, Carlos Scheidegger

Abstract—The data cube is a convenient abstraction for exploring multi-dimensional datasets. However, exploring large datasets as data cubes at interactive speeds requires specialized data structures and systems. Such systems exist, but there is no comprehensive comparison of these systems on realistic data. We present a benchmark that uses real-world and synthetic spatiotemporal data to explore the performance of these systems on realistic query sets. We compare four existing systems using our implementation and demonstrate performance differences between them. Finally, we give a set of guidelines for choosing a system for applications.

Index Terms—Benchmark, Data Cube

1 Introduction

Data cubes are a useful tool for exploring aggregates of high dimensional data [**?**]. However, executing queries on data cubes of large datasets fast enough to support an interactive user interface is technically difficult because current systems use excessive memory usage or demand a high degree of parallelism [**?**, **?**].

To address these problems, systems such as *Nanocubes* and *Hashedcubes* allow fast queries in several dimensions by storing the dataset in a server-side structure on dedicated hardware [**?**][**?**]. This approach is much more complex and resources intensive than simpler approaches such as linear scans on a client, as in *datavore* and *crossfilter* [**?**, **?**]. On the other hand, it allows interactive exploration of data sets of tens of millions of points. Alternatively, appropriate limits on the resolution or dimensionality of the dataset can reduce query latency to interactive rates, as is done in *imMens* [3].

Each of these systems were published with performance information exercising the system on real-world test data. However, the published performance measures are not, in general, comparable. Different datasets were used, and critical parameters such as field resolution differ. As such, there is no complete picture of data/schema configurations each of these systems is best suited for, nor how each scales in different configurations.

This paper contributes a benchmark that allows direct comparisons of the query performance of these systems on realistic data. Additionally, we describe an implementation for nanocubes and hashedcubes. Finally, we give guidance for choosing between and configuring these systems based on dataset and application.

2 Related work

Large data visualization frequently requires computing aggregations over the entirety of a dataset to produce counts, histograms, and density plots [**?**]. For an interactive visualization, supporting arbitrary aggregations smoothly on large datasets presents a problem due to performance. To maintain an interactive user experience, a plot of aggregates should refresh with low latency (This is not merely an aesthetic concern; high latency induces large changes in a user's exploration to the detriment of analysis, as explored in [2] [Inline citation here?]). However, naive techniques for computing aggregations, namely linear scans over the dataset, do not scale to datasets with tens of millions of entries.

In this paper we are specifically interested in systems that allow fast queries on large spatiotemporal datasets. Specifically, we mean systems that efficiently support data with the following attributes:

- 1 or more *spatial* attributes, defined as a pair of related coordinates from an ordered domain, often latitude and longitude.
- 0 or more *categorical* attributes, defined as a value from a discrete domain.
- 1 *time* attribute, defined as an attribute over an ordered domain.

2.1 Existing Systems

Several solutions exist for this problem. Nanocubes builds a hierarchical tree of agregations that fits in main memory, in contrast to traditional relational database data cube aggregations which must be stored on disk [**?**]. This system provides query performance that is fast enough for interactive applications and suports queries at different resolutions (up to the limits of the underlying data), but is memory intensive.

A new alternative to nanocubes is hashedcubes. This system achieves similar performance for many common queries but consumes less memory by using a single sorted array, at the cost of poor performance in some cases. Additionally, it does not allow fine queries in sparsely populated areas, as a tradeoff to prevent poor performance in other areas.

ImMens pre-computes and compactly stores aggregates as so-called "data-tiles" [3]. These tiles can be quickly queried using the GPU on the client. ImMens achieves very fast response time (limited by the GPU refresh rate) and does not require a dedicated server, but requires pre-computing data tiles and limits on the resolution and dimensionality of the underlying data.

An alternate solution to this performance challenge is sampling. *BlinkDB* uses parallelism and intelligent sampling strategies to achieve approximate query results with $2-10\%$ error and response times on the order of 2 seconds [1]. This system is designed for datasets on the order of terabytes in size, but does not provide response times fast enough for an interactive application [In the sense we are describing – should specify this more carefully]. For datasets with tens to hundreds of millions of points, the previously described systems can produce much more accurate results using less time and resources. For this reason, we will not benchmark sampling systems here. [This hardly seems like a comprehensive treatment of sampling systems – need to look at more references]

Finally, for small datasets with $1-5$ million elements, it is possible to do queries at near interactive latencies in a web browser. Systems such as *datavore* and *crossfilter* take this approach [**?**, **?**]. While this approach does not scale to tens and hundreds of millions of data points due to latency and bandwidth consumption, it is a simple solution for small datasets.

In this paper we will describe a benchmark aimed at nanocubes, hashedcubes, imMens, and crossfilter/datavore. Despite the performance measurements reported in publications for these systems, it is not possible to definitively state how each of these systems perform relative to each other. It is similarly not well-understood how each of these systems scale with dataset size, dimensionality, and query

- *Martha Stewart is with Martha Stewart Enterprises at Microsoft Research. E-mail: martha.stewart@marthastewart.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.
Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/

distribution. Our contribution is to rectify this situation and provide guidance for choosing between these systems for applications.

[Why are we not covering streaming]

[Why is TPC not applicable? On-disk, not specifically spatiotemporal]

[”Implementing Data Cubes Efficiently” should be cited]

3 Benchmark

[How precise should thie specification be?]

We have the following goals for a benchmark:

- **Use realistic data and queries.** The systems under test should operate on of datasets representative of real-world data. This means datasets of large scale and with varied schema that show up in real-world application. Specifically, the benchmark must test datasets that range from $1-100$ million data points, with 3 or more dimensions [Why is this the right range? What evidence do we have for this?][**?**]. Additionally, the queries run against the systems under test should be realistic. They should reflect queries that result from supporting actual user behavior.
- **Exercise entire data-query space.** The set of configurations and query sets used by the benchmark should span the full range of data sets for which these systems may be reasonably applied.
- **Efficiently cover data-query space.** The set of possible systems, configurations, datasets, and queries is very large. Pragmatically, it is important to choose tests that complete our goals without requiring excessive time or resources to complete.

3.1 Realistic data and queries

In the interest of supporting further research and consistent performance measurements across systems, part of the benchmark will be based on publicly-available data sets. However, we will supplement this with synthetic data when there is no suitable public dataset for testing a particular aspect of performance.

We identify the following categories of datasets as distinct categories for which we expect significant deviation in performance between the systems under test. [Why?]

- 1 Million data points, $3-32$ dimensions. Examples: Splom, Anything
- $5-10$ Million data points, $3-10$ dimensions. Examples: Brightkite, Gowalla, Splom
- 100+ Million data points, $3-5$ dimensions. Examples: Airline, Splom
- A dataset with more than one spatial attribute [Taxicab?].

  [Is a spatial attribute 1 or 2 dimensions]

The first three classes of dataset consist of one spatial attribute, 1 temporal attribute, and $0-29$ categorical attributes. Conservatively, supposing that categorical dimensions have small domains of size k, it is clear that it is unreasonable to exhaustively search up to k^{30} combinations of categorical variables; the problem is exacerbated when we consider that we would ideally test each of those combinations against many different spatial and time ranges. In these cases, we will show that a regular but sparse sampling of the categorical variable combinations will yield sufficient information about the performance characteristics of the systems under test. [Should consider trying to apply low-discrepancy sequences here]

For smaller numbers of dimensions, it will be possible to exhaustively check all combinations of categorical variables [Why would we not sample anyway to make it faster?].

[Realistic benchmark given user sessions, best guess]

[Measuring: memory, query time, build time,]

[Which data sources and why]

[Which queries and why (synthetic and from user sessions)]

4 Experiments

[Which systems (nanocubes, hashedcubes, immens, crossfilter, datavore)]

[Plots, details of experiments (compiling, machine, how many times, more detail is better)]

[Interpretation of results]

5 Discussion (of paper; what did we not do?)

[how to improve this]

[what did we miss]

[Different interface/system induces different user behavior (cite Jeff Heer, The effects of interactive latency on explorative visual analysis)]

[Can be extended to new data, hopefully new queries]

6 Conclusion and Future Work

[What this means for making new systems (easier, negative space)]

[What next (making things parameterizable, what is the next research question)]

Acknowledgments

The authors wish to thank A, B, C. This work was supported in part by a grant from XYZ.

References

[1] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. June 19 2012.

[2] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2014.

[3] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013.