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* [1].

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 [3] [FIXME: 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.

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." 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 [2]. 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 [FIXME: 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. [FIXME: 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 distribution. Our contribution is to rectify this situation and provide guidance for choosing between these systems for applications.

[TODO: Why are we not covering streaming]

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

[TODO: "Implementing Data Cubes Efficiently" should be cited]

• *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/

3 Benchmark

[TODO: Realistic benchmark given user sessions, best guess]

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

[TODO: Which data sources and why]

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

4 Experiments

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

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

[TODO: Interpretation of results]

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

[TODO: how to improve this]

[TODO: what did we miss]

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

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

6 Conclusion and Future Work

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

[TODO: 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]

[2]

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