A Benchmark for Interactive Data Cubes

Sean Stephens, Carlos Scheidegger

Abstract— The data cube is a convenient abstraction for exploring multi-dimensional datasets and aggregations of those datasets over arbitrary attributes. However, representations of large datasets as data cubes tend to consume large amounts of memory, especially as the dimensionality of the data increases. Recently published methods allow exploration of large datasets through the data cube abstraction at interactive speeds by maintaining keeping the data cube representation in main memory. There is no comprehensive comparison of the performance of these systems, and the performance characteristics of each remain poorly understood. We present a benchmark that uses real-world spatiotemporal datasets as well as synthetic data to explore the performance of these systems on real-world and synthetic query sets. We compare several existing systems using an implementation of these benchmarks and demonstrate performance differences between them. We report startup and query latency as well as memory and bandwidth use, where applicable. 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, as they support [Slicing, dicing – find proper names for these operations][**?**]. However, executing queries on data cubes constructed from large datasets of high dimensionality fast enough to support an interactive user interface is technically difficult because current systems use large amounts of memory or demand a high degree of parallelism [**?**, **?**].

To address these problems, recent 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 more complex and resource 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, limits on the resolution or dimensionality of the dataset can allow pre-aggregation techniques that reduce query latency to interactive rates, as in *imMens* [3].

Each of these systems were published with performance data exercising the system on real-world 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 which data and schemas 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. We describe an implementation for nanocubes hashedcubes, imMens, crossfilter/datavore [We really only need one of these – assuming that they utilize similar techniques, showing both really won't add much information], and report performance in terms of startup and query latency, memory consumption, and bandwidth requirements. 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 significant changes in a user's exploration to the detriment of analysis, as explored in [2]). 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 most interested in systems that allow fast queries on data cubes built from large spatiotemporal datasets. Specifically, we mean systems that efficiently support queries with latency ideally less than 100 ms but up to 1 second (using guidance from [2] on data with the following types of attributes:

- 1 or more *spatial* attributes. This type is inspired by geographical location, usually latitude and longitude, but more abstractly is any pair of related coordinates from an ordered domain. Indeed, while some systems provide explicit support for latitude and longitude, others allow arbitrary domains for spatial attributes and leave the choice of encoding to the user of the system.
- 0 or more *categorical* attributes. Abstractly, this means a value from a discrete domain, but as a concrete example might be device type, language, or day of week for a telecommunication dataset.
- 1 *time* attribute. This is any attribute over an ordered domain, but is frequently time.

2.1 Existing Systems

Several solutions exist for this problem. Nanocubes builds a hierarchical tree of agregations that takes advantage of redundancy in the dataset to fit 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.

An alternative to nanocubes is hashedcubes. Hashedcubes achieves similar performance for many common queries but consumes less memory by using a single sorted array representation of the data, at the cost of poor performance in some cases. Additionally, it does not allow fine queries in sparsely populated areas, as a tradeoff to guard against poor performance pathological queries.

ImMens pre-computes compact aggregates in 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 an active server [It has a dedicated server– need to verify that it is basically just file serving], but requires both

- *Sean Stephens is at the University of Arizona. E-mail: seanastephens@email.arizona.edu*
- *Carlos Scheidegger is at the University of Arizona. E-mail: cscheid@email.arizona.edu [Check this]*

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/

pre-computing data tiles and transmitting them to a client, as well as imposes limits on the resolution and dimensionality of the client's view into the dataset.

For moderate datasets with $1-5$ million elements, it is possible to do queries at 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, and has scales well as dimensionality increases for fixed dataset size.

As an alternative to exact representations, some systems increase performance on large datasets by sampling. *BlinkDB* uses parallelism and good choice of sampling strategy 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 by our definition here. In a related technique, systems like *[SAMPLING SYSTEM]* stream data to a client interface with low initial latency, but potentially several seconds until sufficient accuracy for common decision problems [**?**]

["Implementing Data Cubes Efficiently" might be cited here...]

For datasets with tens to hundreds of millions of points, nanocubes, hashedcubes, imMens, and datavore can produce exact [is exact an overstatement here?]results using less time then these streaming/sampling systems. Furthermore, there are fundamental differences in the visualizations that sampling/streaming systems and exact systems support [**?**]. For this reason, we leave a benchmark comparing sampling/streaming systems to future work, and make no claim about their suitability in application compared to exact systems.

In this paper we will describe a benchmark aimed at nanocubes, hashedcubes, imMens, and [crossfilter/datavore]. Some of these systems have published performance measurements. Despite this, 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 and dimensionality. Our contribution is to rectify this and provide guidance for choosing between these systems for applications.

[Why is TPC not applicable? Built for on-disk systems, not specifically spatiotemporal]

3 Benchmark

We have the following goals for a benchmark:

- **Use realistic data and queries.** The systems under test should operate on datasets representative of real-world data. This means large datasets with varied schema that frequently occur in real-world application. The benchmark should 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, and ideally be derived from queries that result from 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, and completely exercise the system on each such dataset.

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.

Publicly available datasets fall into several categories in terms of schema and size:

[Formatting]

[Explain why these are representative, good, etc.]

Table 1. Dataset categories

Name	Size	Dimensions	Datasets
Moderate	1-10M	1 spatial, 1 time, 0-7 categorical	Brightkite, Gowalla, Synth
Large	10-150M+	1 spatial, 1 time, 0-4 categorical	Airline, Synthetic
Multi-spatial	10-100M	2 spatial, 1 time, 0-2 categorical	Airline, taxicab, syntheti

Our query sets are a combination of query sequences collected anonymously from user interactions with applications supported by nanocubes, as well synthetic query sequey.

[Explain why user traces are representative of query patterns, with a massive caveat because they were collected through a small variety of interfaces]

We use query sequences collected from users because they are, at the very least, an approximation of possible user behavior. [Make this caveat stand out more:]A major weakness of this source of data is that the user interactions occurred in the context of a specific user interface. It is well-established that the design of a user interface greatly impacts the patterns of exploration of users [Why][**?**]. At the very least, an interface without a mechanism to express a particular query will never allow that query in a user session. On the other hand, using actual user data provides important advantages over purely synthetic query sets. Indeed, actual user data may have different query types in realistic proportions. Those proportions would be difficult to ascertain without examining user data. Additionally, the order of the queries is potentially important. Ordering may have complex interactions with caching at hardware and software levels as well as software optimizations in servers or web browsers. In the absence of a better model in which to properly understand the interactions of these factors, using realistic query sequences gives us a reasonable starting place to understand the performance of these systems.

[Include screenshots of nanocube interfaces]

To augment collected query sequences, we will generate an additional list of queries. Intuitively, this additional query set will be the product of

1. A sample of square spatial queries at minimal resolution, a sample of thin rectangular spatial queries at minimal resolution, a sample of square spatial queries at high resolution, and a sample of thin rectangular spatial queries at high resolution

2. A sample of combinations of categorical dimensions

3. A sample of large time ranges, a sample of short time ranges.

For the multi-spatial dataset, (1) will be applied twice.

In contrast to user-generated queries, the synthetic queries are less realistic but potentially cover a broader range of possible queries. Thus, we use the user queries to measure performance under realistic load, and the synthetic queries to explore the rest of the query space for each system.

[Should consider trying to apply low-discrepancy sequences here]

[Describe what we want to measure; memory, query time, build time]

The first metrics of interest during queries will be query latency and memory consumption by the server. For nanocubes and hashedcubes this means memory consumed on the server-side, while for [crossfilter/datavore]and imMens it means browser/GPU memory consumption. Of secondary interest is startup time. For nanocubes and hashedcubes this means the time that the server takes to read in data and be ready to accept queries. For imMens and [crossfilter/datavore]this means the bandwidth/time required for the initial data transfer to the client. We also report tile construction time for imMens. [This all needs to be torn out and re-written]

4 Experiments

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

5 Discussion

[How to improve the benchmark itself]
[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

[Ask Carlos]

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.