CIS 602-01: Scalable Data Analysis

Data Cubes

Dr. David Koop
CAP Theorem

Scalability: CAP Theorem

Remains accessible and operational at all times.

Availability

CA

Traditional relational databases: PostgreSQL, MySQL, etc.

Pick Two!

AP

Voldemort, Riak, Cassandra, CouchDB, Dynamo-like systems

CP

HBase
MongoDB
Redis
Memcached
BigTable-like systems

Partition Tolerance

Only a total network failure can cause the system to respond incorrectly.

Consistency

Commits are atomic across the entire distributed system.

[E. Brewer]
Google Cloud Spanner: NewSQL

<table>
<thead>
<tr>
<th></th>
<th>CLOUD SPANNER</th>
<th>TRADITIONAL RELATIONAL</th>
<th>TRADITIONAL NON-RELATIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>✓ Yes</td>
<td>✓ Yes</td>
<td>✗ No</td>
</tr>
<tr>
<td>SQL</td>
<td>✓ Yes</td>
<td>✓ Yes</td>
<td>✗ No</td>
</tr>
<tr>
<td>Consistency</td>
<td>✓ Strong</td>
<td>✓ Strong</td>
<td>✗ Eventual</td>
</tr>
<tr>
<td>Availability</td>
<td>✓ High</td>
<td>✗ Failover</td>
<td>✓ High</td>
</tr>
<tr>
<td>Scalability</td>
<td>✓ Horizontal</td>
<td>✗ Vertical</td>
<td>✓ Horizontal</td>
</tr>
<tr>
<td>Replication</td>
<td>✓ Automatic</td>
<td>✓ Configurable</td>
<td>✓ Configurable</td>
</tr>
</tbody>
</table>

[https://cloud.google.com/spanner/](https://cloud.google.com/spanner/)
Spanner: Globally-Distributed Database

![Spanner server organization diagram]

**Zone 1**
- zonemaster
- location proxy
- spanserver

**Zone 2**
- zonemaster
- location proxy
- spanserver

**Zone N**
- zonemaster
- location proxy
- spanserver

---

[Corbett et al., 2012]
Interleaved Schema

CREATE TABLE Users {
    uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
    uid INT64 NOT NULL, aid INT64 NOT NULL, name STRING
} PRIMARY KEY (uid, aid),
INTERLEAVE IN PARENT Users ON DELETE CASCADE;

[Corbett et al., 2012]
TrueTime

- API to try to keep computers on a globally-consistent clock
- Uses GPS and Atomic Clocks!
- Time masters per datacenter (usually with GPS)
- Each machine runs a timeslave daemon
- Armageddon masters have atomic clocks
- API:

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TT.now()$</td>
<td>$TT.interval: [earliest, latest]$</td>
</tr>
<tr>
<td>$TT.after(t)$</td>
<td>true if $t$ has definitely passed</td>
</tr>
<tr>
<td>$TT.before(t)$</td>
<td>true if $t$ has definitely not arrived</td>
</tr>
</tbody>
</table>

[Corbett et al., 2012]
Distribution of TrueTime Epsilons

The data shows that these two factors in determining the increase in tail latencies begin to outweigh the improvements in network performance and the reduction in transient network-link congestion. The impact of these factors is most evident in the 99.9th percentile, which represents the performance of the system in the worst-case scenario. The reduction in tail latencies began on March 30 due to networking improvements, but the 99.9th percentile values, sampled right after timeslave daemon polls the time masters, are generally not a problem. However, there can be significant tail-latency issues that cause transient performance degradation.

Table 5 illustrates the distribution of the number of directory-fragment counts in F1. The dataset is tens of terabytes, which is small compared to some other systems. The F1 team has manually sharded many ways, and the uncompressed dataset is too complex to do regularly as a result. The team had to limit growth on the MySQL database by storing some fixed shard. This layout enabled the use of indexes and secondary indexes: writes to more than a few fragments are generally not a problem. However, there can be significant tail-latency issues that cause transient performance degradation.

The MySQL sharding scheme assigned each customer and all related data to a fixed shard. This layout enabled the use of indexes and secondary indexes: writes to more than a few fragments are generally not a problem. However, there can be significant tail-latency issues that cause transient performance degradation.

The reduction in tail latencies began on March 30 due to networking improvements, but the 99.9th percentile values, sampled right after timeslave daemon polls the time masters, are generally not a problem. However, there can be significant tail-latency issues that cause transient performance degradation.

Figure 6: Distribution of TrueTime Epsilons

[Corbett et al., 2012]
Latency: Spanner vs. MySQL

Latency at 3,000 Queries per Second

[P. Bakkum and D. Cepeda, 2017]
Latency: Spanner vs. MySQL

[Latency at 9,000 Queries per Second]

Median Latency (ms)

Query

[Graph showing the comparison between Spanner and MySQL at 9,000 queries per second]

[P. Bakkum and D. Cepeda, 2017]
Max Throughput vs. Nodes

[Graph showing the relationship between Max Throughput and Nodes, with data points at 10 nodes: 10,400, 15 nodes: 13,600, 20 nodes: 17,400, 25 nodes: 25,800, 30 nodes: 33,200.]

[P. Bakkum and D. Cepeda, 2017]
Assignment 3

- Analysis of the IRS Form 990 Filings Data
- Sign up for AWS Educate
- Part 1: Run Spark Locally over Index
  - Process the index file (CSV), organize by return_type
  - Find top dates and months for filings
- Part 2: Run Spark Locally over Subset
  - Find GrossReceiptsAmt max/min/mean for IRS 990EZ filing (XML)
  - Create a histogram of change in net assets/balances
- Part 3: Run Spark on AWS over Full Dataset (Year)
  - (Same queries as above)
  - Check your code locally first
Test 2 and Project Proposals

• Test 2: Nov. 16

• Project Proposals:
  - Identify dataset or project and provide detailed writeup
  - Must have scalability concerns and those must be described
  - AWS Public Datasets
  - Awesome Public Datasets
  - Specific Datasets:
    • AIS Data
    • NBA Data
    • IRS 990 Data
  - Due Nov. 20
Project Proposal: Dataset Analysis

• Description of dataset(s). Include the URL(s). If a dataset is not available online, please describe how you have access to it.

• Existing work. List papers or Web articles with existing analyses of the dataset(s) or similar datasets. (at least 3)

• Questions. List questions you would like to answer about the data. (at least 3) Be specific. You are not obligated to answer every question you list here for the project.

• Techniques. List the techniques you plan to use to answer the questions. Be specific. "Visualize the data" is not specific. "Create a choropleth map showing each county colored by incidence of disease" is specific.

• Scalability. Be specific about what scalability challenges you face with the dataset
Project Proposal: Research Project

• Description of the problem. Describe the problem at a high-level first, then add any relevant details.

• Existing work. List papers that address the problem or related problems. (at least 3)

• Scalability Challenges: What are the scalability challenges you face with this particular problem?

• Ideas. What do you plan to investigate to solve the problem?

• Evaluation. How do you plan to compare your work to the existing solutions to show improvement? What tests do you plan to run?
Data Cubes

J. Han, M. Kamber, and J. Pei
Data Cube: A Lattice of Cuboids

0-D \((apex)\) cuboid

1-D cuboids

2-D cuboids

3-D cuboids

4-D \((base)\) cuboid

[Han et al., 2011]
Dimensions and Measures

• Dimensions: attributes than can be organized as a hierarchy
  - Parents summarize their children
  - Examples: time(day, week, month, quarter, year), location(country, region, state, city, district)

• Measures: A number
  - Examples: number of orders, forecasted revenue
  - Types:
    • Distributive: When result applied to n aggregate values equals the result applied to all values (e.g. count, sum, min, max)
    • Algebraic: When result can be computed by a function on distributive measures (e.g. mean, stddev)
    • Holistic: When no constant bound on storage size needed to store a subaggregate (e.g. median, mode, rank)

[Han et al., 2011]
Cube Operations

• Roll-up: aggregate up the given hierarchy
• Drill-down: refine down the given hierarchy
• Roll-up and drill-down are "inverses"
Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

L. Lins, J. T. Klosowski, and C. Scheidegger
Goal: Interactive Exploration of Data Cubes

Linked view of tweets in San Diego, US
Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

Abstract

Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop’s main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

1 INTRODUCTION

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe’s famous quartet shows, summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization.

Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In
iPhone vs. Android Map
Zoom into Chicago
SuperBowl in Indianapolis

Total count: 1,686,897 of 210,634,624

device
windows
ipad
android
iphone
none

Abstract
— Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop’s main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

Index Terms
— Data cube, Data structures, Interactive exploration

1 INTRODUCTION
As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe’s famous quartet shows [3], summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration [15, 31]. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization [33, 18, 21].

Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In
New Year's Eve in Manhattan

![Map and visualization of New Year's Eve in Manhattan](image)

**Fig. 1.** Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

**Abstract**

Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop's main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

**Index Terms**

Data cube, Data structures, Interactive exploration

---

1 Introduction

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe's famous quartet shows\[3\], summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration\[15, 31\]. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization\[33, 14, 18, 21\].

Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In
Aggregations on Spatiotemporal Data

- Spatial: e.g. counting events in a spatial region (world or San Fran.)
- Temporal: e.g. queries at multiple scares (hour, day, week, month)
- Seek to address Visual Information Seeking Mantra:
  - Overview first, zoom and filter, details-on-demand
- Multidimensional:
  - Latitude, Longitude, Time + more
Data Cube Aggregations

Relation A

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Android</td>
<td>en</td>
</tr>
<tr>
<td>US</td>
<td>iPhone</td>
<td>ru</td>
</tr>
<tr>
<td>South Africa</td>
<td>iPhone</td>
<td>en</td>
</tr>
<tr>
<td>India</td>
<td>Android</td>
<td>en</td>
</tr>
<tr>
<td>Australia</td>
<td>iPhone</td>
<td>en</td>
</tr>
</tbody>
</table>

Aggregation B

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>All</td>
<td>5</td>
</tr>
</tbody>
</table>

Group By on Device, Language C

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Android</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>ru</td>
<td>1</td>
</tr>
</tbody>
</table>

Cube on Device, Language D

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>All</td>
<td>5</td>
</tr>
<tr>
<td>All</td>
<td>Android</td>
<td>All</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>All</td>
<td>3</td>
</tr>
<tr>
<td>All</td>
<td>All</td>
<td>en</td>
<td>4</td>
</tr>
<tr>
<td>All</td>
<td>All</td>
<td>ru</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>ru</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>Android</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>en</td>
<td>2</td>
</tr>
</tbody>
</table>

Equivalent to Group By on all possible subsets of \{Device, Language\}

[Lins et. al, 2013]
Nanocube Queries

- Representing natural language queries as data cube queries

<table>
<thead>
<tr>
<th>Natural language query</th>
<th>s</th>
<th>c</th>
<th>t</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>count of all Delta flights</td>
<td>R</td>
<td>U</td>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>count of all Delta flights in the Midwest</td>
<td>R</td>
<td>Midwest</td>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>count of all flights in 2010</td>
<td>R</td>
<td>U</td>
<td>D</td>
<td>R 2010</td>
</tr>
<tr>
<td>time-series of all United flights in 2009</td>
<td>R</td>
<td>D</td>
<td>R</td>
<td>2009</td>
</tr>
<tr>
<td>heatmap of Delta flights in 2010</td>
<td>D</td>
<td>tile0</td>
<td>R</td>
<td>2010</td>
</tr>
</tbody>
</table>

- s = space, c = category, t = time
- R = rollup, D = drill down
- <value> after RD = subset of dimension's domain, U = universe
- Note that time queries are stored in an array of cumulative counts

[Lins et. al, 2013]
Building a Nanocube

Indexing Schema

\[ S = [\ell_{\text{spatial1}}, \ell_{\text{spatial2}}, \ell_{\text{device}}] \]
Summed-area Table

• Every node in the previous figure stores an array of timestamped counts like this:

![Summed-area Table Diagram]

query/tseries/1/3/4

start at bin 1, use buckets of 3 bins each, and collect 4 of these buckets

solve using...

A Summed Table Sparse Representation for Counts
Building a Nanocube: Step 1

Indexing Schema

\[ S = [[\ell_{\text{spatial1}}, \ell_{\text{spatial2}}], [\ell_{\text{device}}]] \]

[Lins et. al, 2013]
Building a Nanocube: Step 2

Indexing Schema

\[ S = [ [\ell_{spatial1}, \ell_{spatial2}], [\ell_{device}] ] \]

Five Tweets: Location and Device

- \( \ell_{device}(\bigcirc) = \text{Android} \)
- \( \ell_{device}(\bullet) = \text{iPhone} \)

<table>
<thead>
<tr>
<th>( \ell_{spatial1} )</th>
<th>( \ell_{spatial2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0 1,0</td>
<td>00,11 01,11</td>
</tr>
<tr>
<td>0,0 1,1</td>
<td>00,10 01,10</td>
</tr>
<tr>
<td>0,1 1,0</td>
<td>00,01 01,01</td>
</tr>
<tr>
<td>1,0 1,1</td>
<td>00,00 01,00</td>
</tr>
</tbody>
</table>

parent-child (same dimension):
proper | shared
content (next dimension):
proper | shared
updated in current step
dimension boundary

[Lins et. al, 2013]
Building a Nanocube: Step 3

3.

[Diagram showing the process of building a nanocube with nodes and edges labeled with coordinates and device types such as iPhone and Android.]

parent-child (same dimension):
proportional
shared
content (next dimension):
proportional
shared
updated in current step
dimension boundary

Indexing Schema
S = [{spatial1, spatial2}, {device}]

[Lins et. al, 2013]
Building a Nanocube: Step 4

4.

parent-child (same dimension):
proper
shared
content (next dimension):
proper
shared
updated in current step
dimension boundary

Indexing Schema

\[ S = [\{l_{\text{spatial1}}, l_{\text{spatial2}}\}, l_{\text{device}}] \]

[Lins et. al, 2013]
Building a Nanocube: Step 5

5.

5.1. Building the Nanocube: Step 5

- **iPhone**
- **Android**

- **O2** **O3** **O5**
- **O1** **O4**
- **O1** **O2**
- **O3** **O5**
- **O4**

- **parent-child (same dimension):**
  - **proper**
  - **shared**

- **content (next dimension):**
  - **proper**
  - **shared**

- **updated in current step dimension boundary**

Indexing Schema

\[ S = \{ \{e_{\text{spatial1}}, e_{\text{spatial2}}\}, \{e_{\text{device}}\} \} \]

[Lins et. al, 2013]
Nanocubes Discussion

- Save space by organizing the data in a manner that takes advantage of data sparseness
- Limited to one spatial dimension, one temporal dimension
Example: American vs. Delta
Example: Cell Data Records

Fig. 10. Highlights of a visual analysis session of the CDR dataset, with 1,043,884,027 records. We noticed the different patterns in call volume by interacting with the dataset and trying different regions and category selections. Notice the patterns occur at different spatial and temporal scales.

In the following sections, we provide a brief overview of each of the datasets, followed by an overall summary of our experimental results in section 6.8. For each of the experiments, we paid particular attention to how much memory was required to build and store the nanocube index, as well as the overall complexity of the dataset itself, which varied greatly from one to the next. Once the nanocubes were constructed, we queried them using one or both of our front-end clients to highlight the ease with which analysts could explore the data.

The query times and bandwidth usage across all experiments are consistent, so we report them in aggregate here. The mean query time was 800 µs (less than 1 millisecond) with a maximum of 12 milliseconds. The output size per query averaged 5KB, with a maximum size of 50KB (geographical tiles dominated bandwidth usage). Our server currently uses no compression, although we plan to support transparent gzip stream encoding. The mean number of queries for the C++ client was 100 requests per second. The HTML5 client is much quieter, at around 1 query per second, since linked views are only updated when a brush is released. The C++ client was designed for LANs, and its bandwidth usage is around 5Mbps, well within current capacities.

6.1 Twitter

Between November 2011 and June 2012, we collected about 210 million tweets that originated in the United States using Twitter’s public feed which provides a representative sampling of all tweets. The rate of tweets obtained averaged about one million per day. The data was streamed in the form of JSON objects, from which we extracted the following attributes: latitude and longitude of the device, the time the tweet occurred, the client application used, the type of device, and the language of the tweet. The categorical dimensions in our data (application, device, language) had respectively 4, 5, and 15 distinct values. With a nanocube built using this data, we could quickly explore the data to better understand the areas in which one device is more popular than another, where each of the languages is most prevalent, and how that information changes over time (see Figure 7).

6.2 Airline Commercial Flights History

This publicly available dataset contains data for every commercial flight in the United States over a 20 year period (1987-2008). For over 120 million flights, the records include the scheduled departure and arrival times, the actual departure and arrival times, the origin and destination airports, the airline, and other fields. For this experiment, we built our index using the origin airport (for latitude and longitude), scheduled departure time, the departure delay, and the airline. This allows us to answer queries related to overall departure delays for any airports, airlines, time of day, or combinations thereof. In Figure 8 we present an overview on the weekly percentages of total commercial flights in the U.S. for a 20 year period of Delta and American Airlines.
Example: Cell Data Records

Fig. 10. Highlights of a visual analysis session of the CDR dataset, with 1,043,884,027 records. We noticed the different patterns in call volume by interacting with the dataset and trying different regions and category selections. Notice the patterns occur at different spatial and temporal scales.

In the following sections, we provide a brief overview of each of the datasets, followed by an overall summary of our experimental results in section 6.8. For each of the experiments, we paid particular attention to how much memory was required to build and store the nanocube index, as well as the overall complexity of the dataset itself, which varied greatly from one to the next. Once the nanocubes were constructed, we queried them using one or both of our front-end clients to highlight the ease with which analysts could explore the data.

The query times and bandwidth usage across all experiments are consistent, so we report them in aggregate here. The mean query time was 800 µs (less than 1 millisecond) with a maximum of 12 milliseconds. The output size per query averaged 5KB, with a maximum size of 50KB (geographical tiles dominated bandwidth usage). Our server currently uses no compression, although we plan to support transparent gzip stream encoding. The mean number of queries for the C++ client was 100 requests per second. The HTML5 client is much quieter, at around 1 query per second, since linked views are only updated when a brush is released. The C++ client was designed for LANs, and its bandwidth usage is around 5Mbps, well within current capacities.

6.1 Twitter
Between November 2011 and June 2012, we collected about 210 million tweets that originated in the United States using Twitter’s public feed which provides a representative sampling of all tweets. The rate of tweets obtained averaged about one million per day. The data was streamed in the form of JSON objects, from which we extracted the following attributes: latitude and longitude of the device, the time the tweet occurred, the client application used, the type of device, and the language of the tweet. The categorical dimensions in our data (application, device, language) had respectively 4, 5, and 15 distinct values. With a nanocube built using this data, we could quickly explore the data to better understand the areas in which one device is more popular than another, where each of the languages is most prevalent, and how that information changes over time (see Figure 7).

6.2 Airline Commercial Flights History
This publicly available dataset contains data for every commercial flight in the United States over a 20 year period (1987-2008). Over 120 million flights, the records include the scheduled departure and arrival times, the actual departure and arrival times, the origin and destination airports, the airline, and other fields. For this experiment, we built our index using the origin airport (for latitude and longitude), scheduled departure time, the departure delay, and the airline. This allows us to answer queries related to overall departure delays for any airports, airlines, time of day, or combinations thereof. In Figure 8 we present an overview on the weekly percentages of total commercial flights in the U.S. for a 20 year period of Delta and American Airlines.
TopKube: A Rank-Aware Data Cube for Real-Time Exploration of Spatiotemporal Data

F. Miranda, L. Lins, J. T. Klosowski, and C. T. Silva
TopKube: What about Top-k and Rankings?

- Aggregates are interesting
- Also, often interested in top-\(k\) answers given particular criteria
- …or rankings
- Search over time and space but find specific examples
- TopKube is a rank-aware data structure that computes top-\(k\) queries with low latency so interactive exploration is possible
Example: Basketball

- Shots by time, number of points scored, and location on the court

<table>
<thead>
<tr>
<th>team</th>
<th>player</th>
<th>time</th>
<th>pts</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLE</td>
<td>L. James</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>BOS</td>
<td>R. Rondo</td>
<td>5</td>
<td>2</td>
<td>38</td>
<td>26</td>
</tr>
<tr>
<td>CLE</td>
<td>L. James</td>
<td>7</td>
<td>3</td>
<td>42</td>
<td>35</td>
</tr>
</tbody>
</table>

- Query: Ranked list of the 50 players who took the most shots
  
  - SELECT player, count(*) AS shots FROM table GROUP BY player ORDER BY shots DESC LIMIT 50

- Query: Rank the top 50 players by points made:
  
  - SELECT player, sum(pts) AS points FROM table GROUP BY player ORDER BY points DESC LIMIT 50
Ranking by Shot Location

![Heatmap of NBA shot locations](image)

<table>
<thead>
<tr>
<th>Player</th>
<th>Shots</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony Parker</td>
<td>93</td>
</tr>
<tr>
<td>Rasual Butler</td>
<td>84</td>
</tr>
<tr>
<td>Mickael Pietrus</td>
<td>84</td>
</tr>
<tr>
<td>Jeff Green</td>
<td>82</td>
</tr>
<tr>
<td>Jared Dudley</td>
<td>80</td>
</tr>
<tr>
<td>Jason Richardson</td>
<td>80</td>
</tr>
<tr>
<td>Carlos Delfino</td>
<td>76</td>
</tr>
<tr>
<td>George Hill</td>
<td>75</td>
</tr>
<tr>
<td>Shane Battier</td>
<td>72</td>
</tr>
<tr>
<td>Joe Johnson</td>
<td>71</td>
</tr>
<tr>
<td>Matt Barnes</td>
<td>68</td>
</tr>
<tr>
<td>Brandon Rush</td>
<td>67</td>
</tr>
<tr>
<td>Mo Williams</td>
<td>65</td>
</tr>
<tr>
<td>Steve Blake</td>
<td>63</td>
</tr>
<tr>
<td>Arron Afflalo</td>
<td>63</td>
</tr>
<tr>
<td>Charlie Bell</td>
<td>63</td>
</tr>
<tr>
<td>Courtney Lee</td>
<td>62</td>
</tr>
<tr>
<td>Stephen Jackson</td>
<td>61</td>
</tr>
<tr>
<td>Marvin Williams</td>
<td>61</td>
</tr>
<tr>
<td>Ray Allen</td>
<td>60</td>
</tr>
</tbody>
</table>

![Bar chart](image)

[F. Miranda et al., 2017]
Multidimensional Binning Model

• S: Binning Schema
• R: Dataset of Records
• A: Association relation between records and product-bins from schema S

New Key Dimension associated with product bin

\{q, v, \sigma, \Sigma v_i\}

• q_i is the ith smallest key that appears in product bin
• v_i is the value of the measure for key q_i in the product bin
• \sigma_i is the index of the key with its largest value
Space and Time Bin Hierarchies

**Additive measures** can naturally count occurrences (e.g., how often an event occurs, just sum the vectors). In this 3D measure example, it is used to illustrate how additive measures can be derived. We do not deal with post-processed measures. Instead, we focus on simple additive scalar measures derived by post-processing an additive measure \[32\].

Correlations can also be derived from additive measures. For example, we might be interested in the spatial and temporal correlations of shot lengths, which can be obtained by combining the measure values of product-bins related to these regions.

In general, one cannot derive the measure value of the union of two sets of values by knowing the median of each set. We avoid this problem here by restricting our measure model to be additive. We can use the union of two sets of values by combining the measure values of product-bins that are semantically meaningful.

For sets of product-bins that are semantically meaningful, we can use the union of two sets of values by combining the measure values of product-bins that are semantically meaningful. In addition to scalars, we can also use additive measures to derive the mean and variance of weights for any set of product-bins.

A Nanocubes-like approach can efficiently retrieve a measure model \[M\] from a potentially large set of buckets. For example, “Who are actually interested in identifying the top-10 players that make the most shots in the NBA?”

To find out, for each player associated with a shot in the game, we can use a spatial and temporal bin hierarchy to quickly access the measure value. This allows us to report the top-20 players that make the most shots from the potentially large set of shots.

Space and Time Bin Hierarchies are useful for many applications, but are especially useful for interactive applications where quick access to measure values is required. For instance, in a barchart, a pixel in a heatmap is associated with one number to any product-bin over several dimensions, the particular mapping \(\mu\) is a quad-tree hierarchy: here we show a 624 bin selection with a measure \(M\) and its product-bins \(B\) that encodes a cube relational operator \(\mu\). The idea of precomputing and dividing second entry by first entry is a relational operator used to compute measures. The remainder of this paper assumes simple additive scalar measures. We do not deal with post-processed additive measures.

Since there is no ranking information encoded in a measure model, it is a hard hit in interesting use cases, analogous to the use of a bar chart. However, instead of simply counting occurrences, we can use additive measures to derive the mean and variance of weights for any set of product-bins.
Example: One Spatial Dim. and A,B,C events
Three Algorithms

- **Sweep:** Use a priority queue where the product bin with the current smallest key is on the top

- **Threshold:** don't do a full scan, use extra information about ranking

- **Hybrid:**
  - Threshold has best theoretical guarantee but some sparse cases can be faster
  - Use Sweep on small input lists, Threshold on denser problem
Top-edited Wikipages in Nevada and Mississippi

![Map of Wikipages edits in Nevada and Mississippi](image)

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baton Rouge, Louisiana</td>
<td>323</td>
</tr>
<tr>
<td>University of Mississippi...</td>
<td>250</td>
</tr>
<tr>
<td>Mississippi</td>
<td>216</td>
</tr>
<tr>
<td>Jackson, Mississippi</td>
<td>206</td>
</tr>
<tr>
<td>Louisiana State University...</td>
<td>189</td>
</tr>
<tr>
<td>Mississippi State University...</td>
<td>169</td>
</tr>
<tr>
<td>WVLA-TV</td>
<td>158</td>
</tr>
<tr>
<td>Ole Miss Rebels football...</td>
<td>155</td>
</tr>
<tr>
<td>List of Star Wars books...</td>
<td>131</td>
</tr>
<tr>
<td>Louisiana</td>
<td>122</td>
</tr>
<tr>
<td>New Orleans Saints</td>
<td>107</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reno, Nevada</td>
<td>303</td>
</tr>
<tr>
<td>Early Christianity</td>
<td>284</td>
</tr>
<tr>
<td>Comparison of the AK-47 and M1...</td>
<td>273</td>
</tr>
<tr>
<td>Las Vegas Academy</td>
<td>225</td>
</tr>
<tr>
<td>Timeline of Christianity...</td>
<td>216</td>
</tr>
<tr>
<td>Las Vegas</td>
<td>204</td>
</tr>
<tr>
<td>Council of Jerusalem</td>
<td>192</td>
</tr>
<tr>
<td>Paul the Apostle</td>
<td>190</td>
</tr>
<tr>
<td>University of Nevada, Las Vegas...</td>
<td>189</td>
</tr>
<tr>
<td>Nevada</td>
<td>188</td>
</tr>
<tr>
<td>Antinomianism</td>
<td>188</td>
</tr>
</tbody>
</table>
Geolocated Flickr tags in Africa

![Geolocated Flickr tags in Africa](image)

- **africa**: 652
- **namibia**: 275
- **afrique**: 258
- **senegal**: 253
- **nigeria**: 198
- **west_africa**: 156
- **square**: 125
- **iphoneography**: 124
- **square_format**: 123
- **instagram_app**: 123
- **dakar**: 122

<table>
<thead>
<tr>
<th>Hashtag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>africa</td>
<td>7,051</td>
</tr>
<tr>
<td>freewheely.com</td>
<td>6,816</td>
</tr>
<tr>
<td>bicycle</td>
<td>6,147</td>
</tr>
<tr>
<td>angola</td>
<td>1,146</td>
</tr>
<tr>
<td>cameroon</td>
<td>917</td>
</tr>
<tr>
<td>cameroun</td>
<td>911</td>
</tr>
<tr>
<td>jb</td>
<td>783</td>
</tr>
<tr>
<td>gabon</td>
<td>779</td>
</tr>
<tr>
<td>roads</td>
<td>536</td>
</tr>
<tr>
<td>ghana</td>
<td>518</td>
</tr>
<tr>
<td>senegal</td>
<td>500</td>
</tr>
</tbody>
</table>

**Fig. 7. Comparing the top edited articles in Nevada and Mississippi.**

**Fig. 8. Geolocated Flickr tags in Africa:**

- **1. Select Paris Area**
- **2. Observe Uncommon Spike on Wed. Jan 7, 2015**
- **3. Select this Spike and Observe Top-10 Hashtags**
- **4. Select Charlie Hebdo’s Top Hashtags and Observe its Temporal Volume Pattern**

**Fig. 9. Microblog exploration using T-KUBE.**
Top Hashtags in Paris related to Charlie Hebdo

1. Select Paris Area


3. Select this Spike and Observe Top-10 Hashtags

4. Select Charlie Hebdo’s Top Hashtags and Observe its Temporal Volume Pattern

1. #jesuischarlie 4,456
2. #charliehebdo 4,190
3. #lrt 1,146
4. #paris 607
5. #gagnetaplace 447
6. #charliehebdo 418
7. #off 402
8. #lt 335
9. #noussommescharlie 197
10. #rip 187
GitHub Top commits near urban centers

![GitHub Top commits near urban centers](image.png)
Evaluation

![Empirical cumulative distributions of the time to retrieve the top-32 valued keys for 100 spatiotemporal queries on the microblog dataset.]

- PostGIS
- Threshold Algorithm
- Sweep
- Hybrid 0.75
- Hybrid 0.50
- Hybrid 0.25

Cumulative Probability

\[ \log_{10}(\text{milliseconds}) \]

- 1ms
- 10ms
- 100ms
- 1s
- 10s
- 100s

- 0%
- 20%
- 40%
- 60%
- 80%
- 100%

The number of entries is the sum of the sizes of the ranks (i.e., the total number of keys in all ranks). Note that number of ranks is the number of lists (of key values) from the selection. Each of the input ranks/all entries consists of one thousand TKR runs on Line 23 in TA are largely wasted effort. If we follow the overall thick solid gray line in the keys column, we see that more than 10% of the problems have 170 or more. If we check the table entry in row, we see that more than 60% involved 100k keys or more. So, given that most problems (more than 60%) involved 100k keys or more, we see that more than 50% of the problems have 170 or more.