CIS 602-01: Scalable Data Analysis

Machine Learning

Dr. David Koop
## OLTP vs. OLAP

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>users</strong></td>
<td>clerk, IT professional</td>
<td>knowledge worker</td>
</tr>
<tr>
<td><strong>function</strong></td>
<td>day to day operations</td>
<td>decision support</td>
</tr>
<tr>
<td><strong>DB design</strong></td>
<td>application-oriented</td>
<td>subject-oriented</td>
</tr>
<tr>
<td><strong>data</strong></td>
<td>current, up-to-date</td>
<td>historical,</td>
</tr>
<tr>
<td></td>
<td>detailed, flat relational</td>
<td>summarized, multidimensional</td>
</tr>
<tr>
<td></td>
<td>isolated</td>
<td>integrated, consolidated</td>
</tr>
<tr>
<td><strong>usage</strong></td>
<td>repetitive</td>
<td>ad-hoc</td>
</tr>
<tr>
<td><strong>access</strong></td>
<td>read/write</td>
<td>lots of scans</td>
</tr>
<tr>
<td></td>
<td>index/hash on prim. key</td>
<td></td>
</tr>
<tr>
<td><strong>unit of work</strong></td>
<td>short, simple transaction</td>
<td>complex query</td>
</tr>
<tr>
<td><strong># records accessed</strong></td>
<td>tens</td>
<td>millions</td>
</tr>
<tr>
<td><strong>#users</strong></td>
<td>thousands</td>
<td>hundreds</td>
</tr>
<tr>
<td><strong>DB size</strong></td>
<td>100MB-GB</td>
<td>100GB-TB</td>
</tr>
<tr>
<td><strong>metric</strong></td>
<td>transaction throughput</td>
<td>query throughput, response</td>
</tr>
</tbody>
</table>

[Han et al., 2011]
From Tables and Spreadsheets to Data Cubes

• A **data warehouse** is based on a multidimensional data model which views data in the form of a data cube

• A **data cube**, such as sales, allows data to be modeled and viewed in multiple dimensions
  
  - **Dimension tables**, such as item (item_name, brand, type), or time(day, week, month, quarter, year)
  
  - **Fact table** contains **measures** (such as dollars_sold) and keys to each of the related dimension tables

• In data warehousing literature, an n-D base cube is called a **base cuboid**. The top most 0-D cuboid, which holds the highest-level of summarization, is called the **apex cuboid**. The lattice of cuboids forms a **data cube**.

[Han et al., 2011]
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]
Data Cube Measures: Three Categories

- **Distributive**: if the result derived by applying the function to n aggregate values is the same as that derived by applying the function on all the data without partitioning
  - E.g., \texttt{count()}, \texttt{sum()}, \texttt{min()}, \texttt{max()}

- **Algebraic**: if it can be computed by an algebraic function with M arguments (where M is a bounded integer), each of which is obtained by applying a distributive aggregate function
  - E.g., \texttt{avg()}, \texttt{min}_N(), \texttt{standard\_deviation()}

- **Holistic**: if there is no constant bound on the storage size needed to describe a subaggregate.
  - E.g., \texttt{median()}, \texttt{mode()}, \texttt{rank()}

[Han et al., 2011]
Multidimensional Data

- Sales volume as a function of product, month, and region

Dimensions: \textit{Product, Location, Time}
Hierarchical summarization paths

[Han et al., 2011]
A Sample Data Cube

Total annual sales of TVs in U.S.A.

[Han et al., 2011]
OLAP Operations

[Han et al., 2011]
Efficient Processing of OLAP Queries

• Determine which operations should be performed on the available cuboids
  - Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection

• Determine which materialized cuboid(s) for OLAP operation:
  - Query: \{brand, province_or_state\} with “year = 2004”
  - 4 materialized cuboids available:
    1. \{year, item_name, city\}
    2. \{year, brand, country\}
    3. \{year, brand, province_or_state\}
    4. \{item_name, province_or_state\} where year = 2004
  - Which should be selected to process the query?

[Han et al., 2011]
NanoCubes: Interactive Data Cube Exploration

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

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 [Lins et al., 2013]...
### 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]
Building a Nanocube

Indexing Schema

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

Fig. 2. An illustration of how to build a nanocube for five points

\[ \ell_{\text{device}}(\Diamond) = \text{Android} \]
\[ \ell_{\text{device}}(\bullet) = \text{iPhone} \]

Five Tweets: Location and Device

1. 2. 3. 4. 5.

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

[Lins et. al, 2013]
TopKube: Rankings

- 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
Example: One Spatial Dim. and A,B,C events

In Figure 3, we show a concrete TUBE data structure with the keys, counts, rank and total levels of a product-bin, to find out if a given key exists in a model is given by a measure which is simply the number of occurrences of a letter in the keys of the key dimension are the letters OP keys OP OP OP OP.

The special dimension in a TUBE consists of one spatial dimension (two level quad-op and it can be modeled as yet another 1-level bin hierarchy, but that special dimension with respect to the additive measure of interest also includes ranking information in the encoding of that dimension.

With this encoding for the key dimension information, in order to quickly solve queries that contain no key or projects in GitHub, or tags in Flickr) and that we are to retrieve ranks of keys, counts, rank and total in the right-most dimension (e.g. players in the NBA example, contains lots of bins (e.g. players in the NBA example, which potentially means a 1,872 product-bin selection.

Clearly, it can be done in any constraints. This query boils down to the single coarsest top ranked keys in a multi-dimensional selection without any constraints. Suppose a user wants the answer of a multi-dimensional selection is not one. For example, in the case of TUBE, in order to quickly solve queries that contain no key or projects in GitHub, or tags in Flickr) and that we are to retrieve ranks of keys, counts, rank and total in the right-most dimension (e.g. players in the NBA example, which potentially means a 1,872 product-bin selection.

In the case of a Nanocube, this extra size is that in order to represent a Nanocube special dimension in involved. Since in all our applications, the answer of a multi-dimensional selection is not one. For example, in the case of TUBE, in order to quickly solve queries that contain no key or projects in GitHub, or tags in Flickr) and that we are to retrieve ranks of keys, counts, rank and total in the right-most dimension (e.g. players in the NBA example, which potentially means a 1,872 product-bin selection.

The advantage here is that now the answer of a multi-dimensional selection is not one. For example, in the case of TUBE, in order to quickly solve queries that contain no key or projects in GitHub, or tags in Flickr) and that we are to retrieve ranks of keys, counts, rank and total in the right-most dimension (e.g. players in the NBA example, which potentially means a 1,872 product-bin selection.

The easiest top-ranked keys in a multi-dimensional selection without any constraints. This query boils down to the single coarsest top ranked keys in a multi-dimensional selection without any constraints. Suppose a user wants the answer of a multi-dimensional selection is not one. For example, in the case of TUBE, in order to quickly solve queries that contain no key or projects in GitHub, or tags in Flickr) and that we are to retrieve ranks of keys, counts, rank and total in the right-most dimension (e.g. players in the NBA example, which potentially means a 1,872 product-bin selection.

Note that efficiently retrieving ranks...
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

[F. Miranda et al., 2017]
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

[F. Miranda et al., 2017]
Assignment 3

- Analysis of the IRS Form 990 Filings Data
- Sign up for AWS Educate
- Part 1: Run Spark Locally over Index
- Part 2: Run Spark Locally over Subset
- Part 3: Run Spark on AWS over Full Dataset (Year)
- Windows Issues:
  - Install Spark separately from conda's pyspark install
  - Run notebook normally and use findspark to link to the pyspark libraries
Test 2

• This Thursday at 5pm
• http://www.cis.umassd.edu/~dkoop/cis602-2017fa/test2.html
• Topics
  - Cloud Computing
  - Cloud Workloads
  - MapReduce
  - Cluster Computing
  - Data Cleaning
  - Data Integration
  - NoSQL Databases
  - NewSQL Databases
  - Data Cubes
Project Proposals

• Due Monday, November 20
• Turn in via myCourses
• 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
  - Taxi Data
Machine Learning

P. Domingos
Scalable Machine Learning

- Tom Augspurger's Post:
  - https://tomaugspurger.github.io/scalable-ml-01.html
  - Think about why you need to scale…
    - Size?
    - Time?
    - Don't forget statistics
  - Using pipelines in scikit-learn
  - Using dask to scale to multiple cores or the cloud
TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems

Google (M. Abadi et al.)
TensorFlow

- TensorFlow is:
  - An **interface** for expressing machine learning algorithms
  - An **implementation** for executing such algorithms
- Runs on lots of different systems
- Google Brain: very-large-scale deep **neural networks**
- Was internal tool, now open-source
- Precursor was DistBelief
- Others: Torch, Caffe, GraphLib, Theano
TensorFlow Programming Model

- Operations and Kernels: abstract computation vs. implementation
- Session: construct graph and run
- Variables: keep state, survive across multiple executions
- Example:

  ```python
  import tensorflow as tf
  
b = tf.Variable(tf.zeros([100]))  # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1))  # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")  # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)  # Relu(Wx+b)
C = [...]  # Cost computed as a function
  ```

  ```python
  s = tf.Session()
  for step in xrange(0, 10):
      input = ...construct 100-D input array ...
      result = s.run(C, feed_dict={x: input})
      print step, result
  ```

Figure 1: Example TensorFlow code fragment

Table 1: Example TensorFlow operation types
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))  # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))  # 784x100 matrix w/ rnd vals
x = tf.placeholder(name="x")  # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)  # Relu(Wx+b)
C = [...]  # Cost computed as a function of Relu

s = tf.Session()
for step in xrange(0, 10):
    input = ... construct 100-D input array ...  # Create 100-d vector for input
    result = s.run(C, feed_dict={x: input})  # Fetch cost, feeding x=input
    print step, result
TensorFlow

• Tensor: think higher-dimensional than scalar, vector
  - Just a matrix

• Devices:
  - Can run on lots of different devices
  - Can specify types of or concrete devices