On the Role of Indexing for Big Data in Scientific Domains

Arie Shoshani
Lawrence Berkeley National Lab
Outline

- Examples of indexing needs in scientific domains
- Scientific Indexing requirements
- Bitmaps indexing as a promising technology
Example of Big Data in Science

Large Hadron Collider: to find the God particle

- **15 PB** per year – sensors capable of 140PB/s
- 27 km tunnel
- ~10,000 superconducting magnets
- Operating temperature 1.9 Kelvin
- Construction cost: US$9 Billion
- Power consumption: ~120 MW

April, 2013
## Typical Event Figures

<table>
<thead>
<tr>
<th>Experiment</th>
<th># members/institutions</th>
<th>Date of first data</th>
<th># events/year</th>
<th>volume/year-TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAR</td>
<td>350/35</td>
<td>2001</td>
<td>$10^8-10^9$</td>
<td>500</td>
</tr>
<tr>
<td>PHENIX</td>
<td>350/35</td>
<td>2001</td>
<td>$10^9$</td>
<td>600</td>
</tr>
<tr>
<td>BABAR</td>
<td>300/30</td>
<td>1999</td>
<td>$10^9$</td>
<td>80</td>
</tr>
<tr>
<td>CLAS</td>
<td>200/40</td>
<td>1997</td>
<td>$10^{10}$</td>
<td>300</td>
</tr>
<tr>
<td>ATLAS</td>
<td>1200/140</td>
<td>2008</td>
<td>$10^{10}$</td>
<td>5000</td>
</tr>
</tbody>
</table>

**STAR:** Solenoidal Tracker At RHIC

**RHIC:** Relativistic Heavy Ion Collider

**LHC:** Large Hadron Collider
Includes: ATLAS, CMS, ...

A mockup of An “event”
What are the indexing challenges?

- Generate large amounts of raw data – referred to as “events”
  - Collected from simulations and experiments

- Post-processing of data
  - Identify elements in data (find particles produced, tracks)
  - Generate summary variables per event
    - e.g. momentum, no. of pions, transverse energy
    - Number of variables is large (50-100)

- Analyze data
  - Use summary variables to characterize events
  - Extract subsets from the large dataset
    - Need to access events based on partial variable specification (range queries)
    - e.g. 
      
      \[
      (0.1 < \text{AVpT} < 0.2) \land (10 < \text{Np} < 20) \lor (\text{N} > 6000)
      \]

- Challenges
  - Search over billions of events
  - Multi-variable search, but only over a subset of the variable
  - Type of query: a needle-in-the-haystack
  - Another type of query: larger subsets for statistical properties
  - Search over numerical values (integers, floating point)
Combustion simulation example

- Combustion simulation: 1000x1000x1000 mesh with 100s of chemical species over 1000s of time steps – $10^{14}$ data values

- This is an image of a single variable (temperature)

- What’s needed is search over multiple variables, such as:
  - Temperature > 1000
  - AND pressure > 106
  - AND HO2 > $10^{-7}$ AND HO2 > $10^{-6}$

- Challenges
  - Multi-variable queries from a subset of variables
  - Search over numerical values
  - Identify large number of regions
Gene functional annotation

Sequence similarity

Gene context provide information for the function of genes.

Functionally related genes are frequently found in the same chromosomal neighborhood.

• Cassette Definition
  • Parallel or Divergent orientation
  • Distance < 300nt
Conserved chromosomal cassettes

- Genes are replaced by protein families (COGs, pfams, IMG ortholog families). One gene → multiple families.
- We refer to these as “properties”, such as “cog0087 cog0088 cog0089 pfam00181 pfam00189 pfam00203 pfam00237”
- Boxes that share two cassettes and two genes, if the genomes are distant phylogenetically (more than species)
- E.g. for black box: blue and red are 1st step relatives.
Why is this problem hard?

- **Size**
  - 100 million cassettes, with properties from about 25,000 possible values (currently).
  - Total number of elements: $2.5 \times 10^{12}$

**Challenge**

- **Query types**
  - Given a cassette find all cassettes that have the same properties in common
    - That is a massive multi-value search
  - Given a cassette find all cassettes that have 2-or-more properties in common (in general k-or-more)
    - Explosive search of all possible combinations of 2-or-more
Big Data Indexing Requirements

- Speed of search
  - Search over billions – trillions data values in seconds

- Multi-variable queries
  - Be efficient for combining results from individual variable search results

- Size of index
  - Index size should be a fraction of original data

- Granularity
  - Ability to produce smaller indexes when granularity can be reduced, such as 1 decimal points, for example

- Parallelism
  - Should be easily partitioned into sections for parallel processing

- Speed of index generation
  - For in situ processing, index should be built at the rate of data generation
Scaling simulations generates a data volume challenge (PBs)
What Can be Done?

- Perform some data analysis and visualization on simulation machine \textit{(in-situ)}
- Reduce Data and prepare data for further analysis \textit{(in-situ)}
Data Analysis

- Two fundamental aspects
  - Pattern matching: Perform analysis tasks for finding known or expected patterns
  - Pattern discovery: Iterative exploratory analysis processes of looking for unknown patterns or features in the data

- Ideas for the analysis of Big Data
  - Perform pattern matching tasks in the simulation machine
    - “In situ” analysis
  - Prepare data for pattern discovery on the simulation machine, and perform analysis on mid-size analysis machine
    - “In-transit” data preparation
    - “Off-line” data analysis
Index:
A Data Structure for Accelerating Data Accesses

- Tree-based indexes
  - E.g. family of B-Trees
  - Commonly used database management systems
  - Sacrifice search efficiency to permit dynamic update

- Multi-dimensional indexes
  - E.g. R-tree, Quad-trees, KD-trees, …
  - Don’t scale for large number of dimensions
  - Are inefficient for partial searches (subset of attributes)

- Hashing
  - Predictable performance
  - Good for locating individual data records

- Bitmap indexes:
  - Good for read-mostly data
  - Handle partial range queries efficiently
  - May have trouble handling data with a large number of distinct values
## Bitmap index for each variable

- Take advantage that index need to be append only
- Generate a bitmap for each possible value of each variable
  - (e.g. for 0<Np<300, have 300 bitmaps)
- Compress each bit vector (some version of run length encoding)
- Need to touch only bitmaps for the specified search

<table>
<thead>
<tr>
<th>variable 1</th>
<th>variable 2</th>
<th>variable n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 0 0 0</td>
<td>0 1 0 0 0</td>
<td>0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0</td>
<td>0 0 1 0 0</td>
<td>0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0</td>
<td>0 0 1 0 0</td>
<td>0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0</td>
<td>0 1 0 0 0</td>
<td>0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>0 0 1 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>0 1 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>0 0 1 0 0</td>
<td>0 0 1 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0</td>
<td>0 0 0 1 0</td>
<td>0 0 0 0 1 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>0 0 0 1 0</td>
<td>1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

...
Basic Bitmap Index

**Data values**

<table>
<thead>
<tr>
<th>Data values</th>
<th>$b_0$</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
<th>$b_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A &lt; 2$</th>
<th>2 &lt; $A$</th>
</tr>
</thead>
</table>

- Easy to build: faster than building B-trees
- Efficient for querying: only bitwise logical operations
  - $A < 2 \rightarrow b_0 \text{ OR } b_1$
  - $A > 2 \rightarrow b_3 \text{ OR } b_4 \text{ OR } b_5$
- Efficient for multi-dimensional queries
  - Use bitwise operations to combine the partial results
- Size: one bit per distinct value per object
  - Definition: Cardinality == number of distinct values
  - Need to control size for high cardinality attributes
- Main idea:
  - highly efficient compression method
  - Compute friendly – can perform operations directly on compressed data
FastBit properties – highly efficient and compact

Main idea:
- Invented specialized compression methods (was patented) that:
  - Can perform logical operations directly on compressed bitmaps
  - Excels in support of multi-variable queries
  - Can partition and merge bitmaps without decompression – essential for parallelization of indexes

FastBit takes advantage of append only data to achieve:
- Search speed by $10x - 100x$ than best known bitmap indexing methods
- On average about $1/3$ of data volume compared to 2-3 times in common indexes because of compression method
- Proven to be theoretically optimal – data search time is proportional to size of the result

Usage
- In multiple scientific application in DOE
- Embedded into in situ frameworks
- Thousands of downloads around the world (open source under source forge), including commercial companies
Methods to Improve Bitmap Index

- **Compression**
  - FastBit compression method: **Word-Aligned Hybrid (WAH) code**
  - 10x speedup over Byte-aligned Bitmap Code

- **Encoding**
  - Multi-level encoding
  - Reduce bitmaps needed for a query
  - 5x speedup

- **Binning**
  - Sometimes we choose to use bins at the fine level to reduce index size
  - Problem: if query falls in the middle of edge bins
  - Solution: **Order-preserving Bin-based Clustering (OrBiC)**
  - 5x speedup for searching bins
Improving Bitmap Indexes: Multi-Level Encoding

- The finest-level may be precise or binned
- Coarse levels are always binned
- Each coarse bin contains a number of fine bins/values
- Queries can be processed with a combination of coarse and fine bitmaps
- Only edge bins need to be resolved at the fine level
- Analysis revealed how to construct the coarse level in order to reduce the query processing time

[Text continues with a diagram showing coarse and fine level encodings and a range query example.]

[Wu, Shoshani and Stockinger 2010]
Two Levels Are Better Than One

- Prove theoretically that the second level needs to have only a small number of bins (15 ~ 50 depending on data)
- Only two levels are necessary
- Result: 5X speedup on average (over WAH compressed 1-level index)

[Wu, Shoshani and Stockinger 2010]
Domain-Specific Challenges – current and future

- Generate index at the rate of data generation in situ
  - Increase level of parallel processing
  - Perform partial index generation per node
  - Take advantage of local NVRAM

- Adapt indexing methods to a variety of data models
  - Irregular grids, geodesic meshes, toroidal meshes
    - How to linearize the space
  - Multi-level grid, such as adaptive-mesh-refinement (AMR)

- Use results of index in subsequent operations
  - Statistical summaries
  - Region growing, region overlaps, …

- Adapt indexing to specialized operations
  - Searches for k-or-more matches
  - Searches based on formulas (plug-in-codes)
Collaboration between SDM and Vis groups

- Use FastBit indexes to efficiently select the most interesting data for visualization

Example: laser wakefield accelerator simulation

- VORPAL produces 2D and 3D simulations of particles in laser wakefield
- Finding and tracking particles with large momentum is key to design the accelerator
- Brute-force algorithm is quadratic (taking 5 minutes on 0.5 mil particles), FastBit time is linear in the number of results (takes 0.3 s, 1000 X speedup)
FastBit adaptation to toroidal meshes

• Extended FastBit indexing capability to search for regions of interest defined on toroidal meshes used for fusion simulations
• Developed algorithms to take full advantages of the regularity present in the magnetic coordinates but not in the Cartesian coordinates
• Much more compact than the general connectivity graph: ~ 200 numbers vs. 6 million numbers
• Labeling query lines using magnetic coordinates is 600-1000 x faster than using connectivity graph
• Developed new Connected Component Labeling algorithm
  – Recently used in Atmospheric Rivers project
Adapting to cassette searches

Results: (1) given a cassette, search all cassettes with the same properties
Done in about **0.07 second** (using vertical bitmaps)

(2) Find similar cassettes with 2 or more properties
Done in about **10-15 seconds** (using horizontal bitmaps)
Flame Front Tracking in Combustion

Challenges

- **Cell identification**
  - Identify all cells that satisfy range conditions

- **Region growing**
  - Connect neighboring cells into regions

- **Region tracking**
  - Track the evolution of the features through time
Big Data Indexing Requirements: Bitmap indexing advantage

- **Speed of search**
  - Search over billions – trillions data values in seconds
    - Yes, with compute-friendly compression

- **Multi-variable queries**
  - Be efficient for combining results from individual variable search results
    - Yes, combining results for each variable as bitmaps is very efficient

- **Size of index**
  - Index size should be a fraction of original data
    - Yes, compression of bitmap index is essential

- **Granularity**
  - Ability to produce smaller indexes when granularity can be reduced, such as 2 decimal points, for example
    - Yes, binning over multiple values proved very effective

- **Parallelism**
  - Should be easily partitioned into sections for parallel processing
    - Yes, if compressed bitmaps can be easily combined (WAH has this property)

- **Speed of index generation**
  - For in situ processing, index should be built at the rate of data generation
    - OK for billions of values, but a trillion value index took 10 minutes (still a challenge)
Architectural Changes that Could Benefit in situ indexing

- NVRAM on each node
  - Can be used to build partial indexes over multiple time steps
  - Can be used to accelerate in-situ index generation

- Take advantage of GPUs
  - Assign index generation for each variable to separate GPUs

- NVRAM between machine and storage system
  - Can be used for generating indexing for post-processing while data is streaming out to be stored on disk
THANKS!

Co-authors:
E. W. Bethel, S. Byna, J. Chou, W. S. Daughton, M. Howison, H. Karimabadi, K.-J. Hsu, K.-W. Lin, V. Markowitz, K. Mavrommatis, Prabhat, A. Romosan, V. Roytershteynz, O. Rübel, A. Shoshani, A. Uselton

FastBit software http://sdm.lbl.gov/fastbit/
FastQuery software http://goo.gl/iBw6V
Scientific Data Management group http://sdm.lbl.gov/