FastBit Indexing
for Searching and Analyzing Massive Data

John Wu
Scientific Data Management
Berkeley Lab

John.Wu@nersc.gov

http://sdm.lbl.gov/fastbit
FastBit Overview

- A bitmap indexing software package that provides extremely efficient search operations over large datasets
  - Measured >10X faster than the most popular bitmap index implementation
  - Contains innovative techniques: efficient compression (patent 2004), multi-level encoding, binning
- Used in many scientific and commercial applications
  - Combustion, astrophysics, network security, drug discovery
- One of 100 most innovative new products in 2008, R&D 100 Award
FastBit Technology 1: Compression

Example: 2015 bits

Main Idea: Use run-length-encoding, but...
partition bits into 31-bit groups [not 32 bit] on 32-bit machines

Merge neighboring groups with identical bits

Name: Word-Aligned Hybrid (WAH) code (US patent)
Key features: WAH is compute-efficient
  ➢ Uses the run-length encoding (simple)
  ➢ Allows operations directly on compressed bitmaps
  ➢ Never breaks any words into smaller pieces during operations
  ➢ Worst case index size 4N words, not N*N (without compression)

Wu, Otoo, and Shoshani 2006
Compressed Index Performance

- WAH compressed indexes are 10X faster than DBMS, 5X faster than our own version of BBC
- Based on 12 most queried variables from a STAR dataset with 2.2 million rows, average column cardinality 222,000
FastBit Technology 2: Multi-Level Encoding

- Prove theoretically that the second level needs to have only a small number of bins (15 ~ 50 depending on the skewness of the data)

- Only two levels are necessary

- Result: 5X speedup on average

- Combined with WAH, could achieve 50X speedup

10X faster

[Wu, Shoshani and Stockinger 2010]
Efficient Numerical Searches

Scientific data contains many variables, multivariate searches are challenging for most techniques, FastBit is very effective for such operations. Application below: locating the flame front in a burning methane jet.

a) CH4 > 0.3

b) temp < 3

c) CH4 > 0.3 AND temp < 3

d) CH4 > 0.3 AND temp < 4

[Stockinger, Wu, Shalf, Bethel 2005]
Efficient Keyword Searches

- FastBit provides efficient indexing techniques for not only numbers, but also text values.
- FastBit can answer queries hundreds of times faster in many cases.
- Test data: Enron email archive.
- Searches involving mixed keywords and numerical values: message contains “California” and sender = “kenneth.lay@enron.com” and date=“2001/07/18”.
- Comparing against MySQL and a version of MonetDB with FastBit.
- More on text searches later by Kamesh Madduri.

[Stockinger, Cieslewicz, Wu, Rotem, Shoshani 2008]
Example application (after data is collected): Forensic Network Data Analysis

- Application scenario: post-incident analysis, looking back into historical records to determine the root cause
- Use network session records produced by BRO intrusion detection system (IDS)
  - Billions of session records available, usually in ASCII text
  - Existing analysis tools can efficiently utilize only a small fraction of the records
- FastBit enables interactive analysis of a large number of records
  - Finding malicious network scans, characterized by a small number of hosts contacting nearly all machines in a network
  - Improving quality of IDS alarms by correlating real-time observations with historical trends
- New features required of FastBit
  - Group-by operator
  - Conditional histogram
Dynamic Histograms In FastBit

- Conditional histograms are common in data analysis
  - E.g., finding the number of malicious network connections in a particular time window
- Top left: a histogram of number of connections to port 5554 of machine in LBNL IP address space (two-horizontal axes), vertical axis is time
  - Two sets of scans are visible as two sheets
- Bottom left: FastBit computes conditional histograms much faster than common data analysis tools
  - 10X faster than ROOT
  - FastBit indexes improve ROOT by 5X

[Stockinger, Bethel, Campbell, Dart, Wu 2006]
Example Application (while data is collected): Real-Time Network Data Analysis

- Application scenario: detect anomalous traffic before it can do any damage
- Existing stream data analysis tool examines current time window only
  - Need to compare current observation with past trends
    - Ex: Host A is contacting many others, is this common in the past? or has this happened in the past?
- Need to do all these in real-time
  - Process current data (efficient stream engine)
  - Archive and index incoming data (efficient index update)
  - Answer queries in archived data (efficient query processing on read-only data)
- New feature required of FastBit: efficient index update
FastBit for Network Traffic Streams

- Working with UCB database group, implemented a prototype system that integrates FastBit with TelegraphCQ, a stream query engine.
- Tested the integrated system with a benchmark of 5 realistic queries.
- Graph above shows that the combined system easily handles 10,000 network sessions per second on a 2.4GHz P4 system.
- A typical desktop computer is sufficient to handle network traffic to a large supercomputer center (~ 500 network sessions per second).

Reiss, Stockinger, Wu, Shoshani, Hellerstein 2007
Summary of FastBit Technology

- FastBit is extremely efficient in many applications: high-energy physics, combustion, astrophysics, network security, drug discovery, ...
- The efficiency comes from new methods and algorithms, careful software engineering, and rigorous theoretical analyses to prove optimality
- Efficient compression for bitmaps
  - Our compression is 10X faster than nearest competitor
  - Proven optimal in computational complexity theory
- Multi-level bitmap encoding
  - Two-level indexes 3-5 times faster than one-level indexes
  - Proven that two levels are sufficient in theory
- Binning for numerical data with a very large number of distinct values
  - Developed a clustering technique that is 3-5 times faster than no binning for high-cardinality data
Overview of FastBit Software

- Task: given a large collection of data, efficiently locate records satisfying a set of conditions
- Example data – structured data:
  - High-energy physics data – billions of collision events, with hundreds of variables
  - Simulation data on a mesh – each mesh point may be viewed as a record/row, each variable a column
- Example queries:
  - Count how many records where pressure > 1000 and temperature between 500 and 1000
  - Select all records where momentum > …
- FastBit solves this search problem with
  - Column data organization
  - Bitmap index
- FastBit is an award-winning open-source software
  - R&D100 award (Wu, Shoshani, Otoo, Stockinger, 2008)
  - Used in a number of research projects
What FastBit Is Not

- Not a database management system (DBMS)
  - It is much closer to BigTable (NoSQL) than to ORACLE
  - Most SQL commands are not supported
- Not a plug-in for a DBMS
  - It is a stand-alone data processing tool
  - No DBMS is needed in order to use FastBit
- Not an internet search engine
  - FastBit is primarily for structured data; internet search engines are for text (unstructured) data
- Not a client-server system
  - We have used FastBit in server programs, but by itself, it is not a client-server system
How Do I Use FastBit

- Command-line tools
  - A handful of command-line tools are available to load data, build indexes, and query data
- Write your own program using FastBit as a library
  - Two levels of API:
    - Class table
    - Class part + query
  - FastBit is written in C++
    - Other languages may access FastBit through C API
Exercise I: Install FastBit Software

- Jun 2011 version: ibis1.2.4
- Unpack fastbit-ibis1.2.4.tar.gz
  - `tar xvzf fastbit-ibis1.2.4.tar.gz && cd fastbit-ibis1.2.4`
- Installation instruction on a Unix-type system
  - Prerequisite – C++ compiler (e.g., g++), pthread library, make, gzip, tar (pretty standard stuff)
  - Commands: `./configure && make –j 2`
- Installation instruction on MS Windows
  - Prerequisite – pthreads-w32, VisualStudio (or another C++ compiler)
  - Compile with VisualStudio
    - Start VisualStudio, open `win/ibis.sln`
  - Compile with MinGW
    - `cd win && make –f MinGW.mak ibis`
- Compilation will take 15 minutes or more
FastBit Data Model

- FastBit is designed to search multi-dimensional append-only data
  - Conceptually in table format
    - rows ➔ objects
    - columns ➔ attributes
- FastBit uses vertical (column-oriented) data organization
  - Efficient for searching
- Physical data layout
  - A data table is split into “partitions”
  - Each partition is a directory in a file system
  - Each directory has a metadata file describing the data partition
  - Each column is represented by a file
BEGIN HEADER
DataSet.Name=testData
Number_of_rows=1000000
Number_of_columns=6
Table_State=1
index = <binning none/> <encoding equality/>
END HEADER

BEGIN Column
name=i9
description=integers 0, 1, ..., and 9
data_type=Int
index = <encoding range/>
END Column
# Basic Bitmap Index

- **First commercial version**
  - Model 204, P. O’Neil, 1987
- **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 row
  - **Definition:** *Cardinality* $==$ number of distinct values
  - Compact for low cardinality attributes, say, cardinality $< 100$
  - Worst case: cardinality $= N$, number of rows; index size: $N \times N$ bits

<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>
Strategies to Improve Bitmap Index

- **Compression**
  - Reduce the size of each individual bitmap
  - Best known compression method: Byte-aligned Bitmap Code [Antoshenkov 1994], used in Oracle bitmap index
  - Word-Aligned Hybrid (WAH) code trades some disk space for much more efficient query processing

- **Encoding**
  - Basic equality encoding, in Model 204
  - Multi-component encoding [Chan and Ioannidis 1998]
  - Multi-level encoding

- **Binning**
  - Equal-width binning, equal-depth binning, …
  - Has to perform candidate check to rule out false positives, time for candidate check dominates the total query response time
  - Order-preserving Bin-based Clustering (OrBiC)
Indexing Option String

- **Syntax**
  - `<binning ... />` `<encoding ... />` `<compression ... />`

- **Binning options**
  - Basic binning option: linear scale, log scale, equal-weight
  - Examples:
    - `<binning none/>`
    - `<binning nbins=1000/>`
    - `<binning begin=10, end=20, scale=linear, nbins=10/>`
    - `<binning precision=2/>`

- **Encoding options**
  - Three basic options: equality, range and interval
  - Combinations:
    - multi-level, e.g., `<encoding interval-equality/>`
    - multi-component, e.g., `<encoding equality ncomp=2/>`

- **Compression options**
  - Public release only supports WAH compression, most users should leave this part out
Indexing Option Suggestions

- Not specifying any option == default option
  - Use the default unless you known something about your data and query
- The following recommendations primarily depends on the column cardinality and the type of query
  - Definition: column cardinality == number of distinct values actually appear in the data partition
- Cardinality < 100:
  - Equality queries: <binning none/> <encoding equality/>
  - Range queries: <binning none/> <encoding interval/>
- Cardinality < 1,000,000 (Nrows/10):
  - Have disk space (index size 2X raw data size):
    <binning none/> <encoding interval-equality/>
- Very high cardinality: <binning none/> <encoding binary/>
- Small number of values to be queried: use them as bin boundaries, treat the number of bins as the column cardinality above
FastBit Command-Line Tools

- All source code for these tools are in examples directory
- Ardea: convert text version of the data records into FastBit raw binary data format – an operation common known as “load”
  - ardea –d output-dir –t text-file –m columnname:typename
- Ibis: query existing data
  - ibis –d data-dir –q “select c1,c2 where c3 > 5 and c4 < 6”
Exercise II: Use Command-Line Tools

- cd tests
- ../examples/ardea -d tmp -m "a:int, b:float, c:short" -t test0.csv
  - ls -l tmp
    total 4
    -rw-r--r-- 1 John Users 400 Jun 18 14:40 -part.txt
    -rw-r--r-- 1 John Users 400 Jun 18 14:40 a
    -rw-r--r-- 1 John Users 400 Jun 18 14:40 b
    -rw-r--r-- 1 John Users 200 Jun 18 14:40 c
- ../examples/ibis -d tmp -build-index "<binning none/>"
- ../examples/ibis -d tmp -q "where a < 5"
- ../examples/ibis -d tmp -q "select a, b, c where a < 5" -v
- More details can be found in doc/quickstart.html, or at http://crd.lbl.gov/~kewu/fastbit/doc/quickstart.html
- Generate synthetic data with tests/setqgen.cpp
- Larger sample data available from http://sdm.lbl.gov/fastbit/data/
Software Layering

- **Abstract view:** `ibis::table` and `ibis::tablex`
  - A table is immutable; to add new records, use `tablex`
  - A query (through function `select`) produces another table
  - Additional functions include: build indexes, get conditional histograms, get column values, …

- **Concrete view:** `ibis::part` and `ibis::query`
  - Each part (partition) is vertically organized
  - An index for a column of a partition is built in memory
  - A query on partition produces a compressed bitmap representing the rows satisfying the specified conditions
Ingesting Data

- Key functions from ibis::tables, used in examples/ardea.cpp
  
```cpp
  ibis::tablex* ta = ibis::tablex::create();

  // create a tablex object
  // parse the metadata string
  ta->parseNamesAndTypes(metadata.c_str());

  // read CSV file, store content in memory
  ierr = ta->readCSV(csvfiles[i], nrpf, del);

  // write the content from memory to the named directory
  ierr = ta->write(outdir, "name", "some description");
```
Simple Queries

- Key functions from ibis::table, used in examples/thula.cpp
  
  ```cpp
  // create a table data object from a directory name
  ibis::table *tbl = ibis::table::create("directory-name");
  
  // a selection forms its own table
  ibis::table *res = tbl->select("select clause", "where clause");
  
  // create a cursor for row-wise access to the results
  ibis::table::cursor *csr = res->createCursor();
  
  // fetch the next row and dump it to std::cout
  while (0 == csr->fetch())
    csr->dump(std::cout);
  ```
Low-Level Query Functions

- Requires the use of `ibis::part` and `ibis::query` (examples/rara.cpp)
  
  ```cpp
  // construct a partition from the given directory
  ibis::part apart(argv[1], static_cast<const char*>(0));
  
  // create a query object with the current user name
  ibis::query aquery(ibis::util::userName(), &apart);
  
  // assign the query conditions as the where clause
  int ierr = aquery.setWhereClause(argv[2]);
  
  // select columns to print
  ierr = aquery.setSelectClause(sel.c_str());
  
  // evaluate the query
  ierr = aquery.evaluate();
  
  // print the selected values
  aquery.printSelected(std::cout);
  ```
Histogram Functions

- Conditional histograms are commonly used in data analyses
  - Count the number of events collected every hour for all events from a particular day (1-D)
  - Count the number of network connection attempts per minute per destination port for a specific duration of time (2-D)
- Class ibis::part also has a set of functions to compute histograms
  - `get1DDistribution`
  - `get2DDistribution`
  - `get3DDistribution`
  - May use regular bins or adaptive bins
  - May be weighted by another variable
- FastBit uses indexes to reduce the amount of data accessed and speeds up the histogram computations
Exercise III: Minimal Query Program

- Write a C++ program that takes a directory name and a query string as arguments, compute the number of records in the directory satisfying the query conditions
- Compile and link
- Example

```cpp
#include <table.h>
int main(int argc, char** argv) {
    ibis::table *tbl = ibis::table::create(argv[1]);
    ibis::table *res = tbl->select(0, argv[2]);
    std::cout << "The number of records satisfying \"" << argv[2] << "\" in "
              << argv[1] << " is " << res->nRows() << std::endl;
    delete res;
    delete tbl;
    return 0;
}
```
Index Sizes to Expect

- Indexes are built for one column and one partition at a time.
- The maximum size of an index is primarily determined by three parameters: the number of rows $N$, the number of bitmaps used $B$, and the bitmap encoding used.
- The range and interval encoded indexes are not compressible in the worst case, therefore their sizes are $N \times B$ bits.
- Under the equality encoding, for a binned index, $B$ is the number of bins, otherwise the number of bitmaps is the number of distinct values (i.e., column cardinality).
  - For small $B$, say, $B < 100$, $N \times B$ bits are needed because bitmaps are likely not compressible.
  - For $B < N / 10$, the common case, index size is about $2N$ words.
- For columns with extremely high cardinality, use binary encoding, which requires $\log B$ bitmaps and $N \times \log B$ bits.
Updating Data and Indexes

- Most efficient way to add new records is to add a partition to an existing table
- Modifying an existing row must be implemented as a deletion following by an append
- Updating an index on a partition will cause a whole new index to be written, which can take a long time compared to the time to answer a query
- To improve response time, such updates are allowed to be delayed, presumably till the system is no longer busy
Parallelism

- Using ibis::part and ibis::query, each parallel processing element could work on one data partition
  - Additional code required to synthesize the final result
- Additional parallelism can come from having each processor answer a part of a query
  - For a query involving “a > 2 and b < 3”, process the condition involving a and b on two separate threads or processors
  - Require additional code to combine the partition results
- Prefer to have more partitions than the number of processors to improve load balancing
- The original version of FastBit was a CORBA server program
  - Current code were the core of the multithreaded server, minus the CORBA functions
  - All existing code is thread-safe
THANKS!

ANY QUESTIONS?

More information at
http://sdm.lbl.gov/fastbit

FastBit mailing list
https://hpcrdrmcg.lbl.gov/cgi-bin/mailman/listinfo/fastbit-users

List of contributors
https://codeforge.lbl.gov/.../AUTHORS