Breaking the Curse of Cardinality on Bitmap Indexes

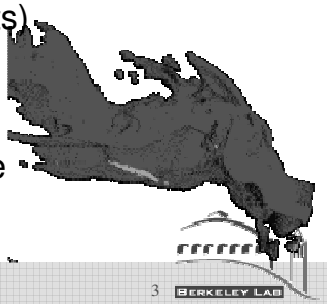
K. John Wu
Kurt Stockinger
Arie Shoshani
Lawrence Berkeley National Lab
University of California
http://sdm.lbl.gov/fastbit/

Outline

1. Achilles Heel of Bitmap Index
2. Order-preserving Bin-based Clustering
3. Analysis
4. Experiment

Problem Definition

- Given a large (static) dataset (data warehouse)
- To answer SQL queries such as
  - Select l_returnflag, …sum(l_quantity) as sum_qty,… From lineitem Where l_shipdate <= date '1998-12-01' - interval '[DELTA]' day (3) group by … [TPC-H Q1]
  - Select cells From Flame-simulation Where temperature > 800 and H_2O_2 concentration > 10^{-6}
- Characteristics:
  - Large datasets: billions of rows, terabytes of base data
  - Typical query may involve many different columns
  - Typical query results may include many rows (hits)
- Objective
  - General: as fast as possible
  - Optimal in computational complexity: O(hits) time



Bitmap Indexes are Efficient for Data Warehouses

Bitmap Index vs. B-tree Index: Which and When?
by Vivek Sharma

always advisable

Understanding the proper application of each index can have a big impact on performance.

Conventional wisdom holds that bitmap indexes are most appropriate for columns having low distinct values—such as GENDER, MARITAL_STATUS, and RELATION. This ass
index is always advisable for systems in which data is
demonstrate here, a bitmap index on a column with 10
efficient as a B-tree index.

IBM Informix Database Design and Implementation Guide

Previous Page | Next Page | Index Dimensional Databases > Implementing a Dimensional Database (XPS) >

Indexes for Data-Warehousing Environments

In addition to conventional (B-tree) indexes, Extended Parallel Server provides the following indexes that you can use to improve ad hoc query performance in data-warehousing environments:

- Bitmap indexes

Optimize Data Warehouse

I know that I am prejudiced on this matter, but I would be ashamed of myself if I were not --Mark Twain

…f a B-tree index. You can use a
…ontain one of only a few values,
…highly duplicate value, a

ontent
ontact
Home

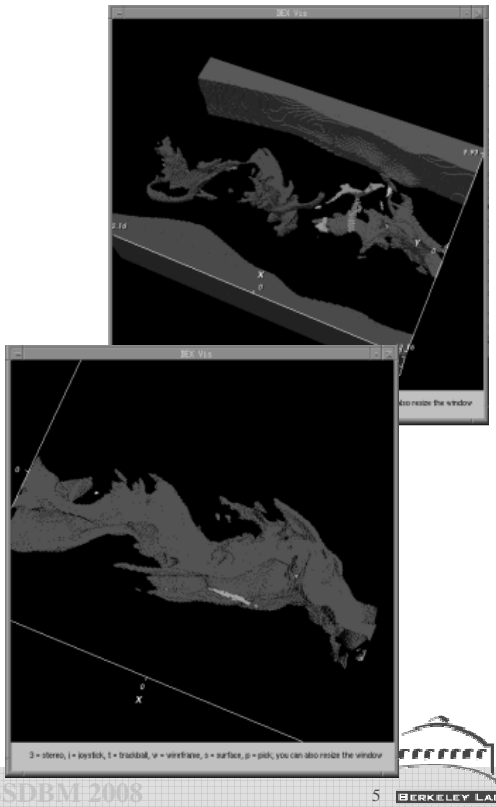
Saturday, Ju
Bitmap
Buy the tick

The star schema and bitmap indexes are a marriage made in heaven.
Jag Singh, VP, JPM Chase

The star schema and bitmap indexes are a marriage made in heaven. The bitmap indexes and their use in accelerating star schema joins is currently available in commercial databases only. Corollary: never buy a database for data warehousing that does not support star schema joins using bitmap indexes.

However, There is a Catch

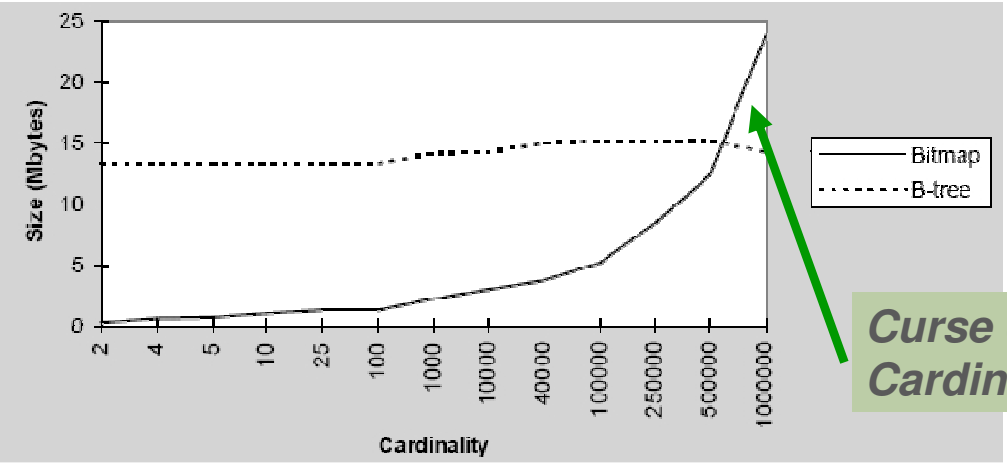
- The efficiency of bitmap indexes decreases as the number of distinct values increases!
- Definition: column cardinality = number of distinct values of a column in a dataset
- As column cardinality increase,
  - The index size increases
  - The query responses time increases



Bitmap Index Size May Be Large

- The size of basic index is proportional to number of distinct values multiplied by number of rows
- …, you should use bitmap indexes on low cardinality columns. On the contrary, a high cardinality field, such as social security number, would not be a good candidate for bitmap indexes.
  - Effective Indexes for Data Warehouses, Roger Deng, DB2 Magazine, Aug. 2004
- Some restrictions on using the bitmap index include: The indexed columns must be of low cardinality—usually with less than 300 distinct values.
  - How and when to use Oracle9i bitmap join indexes, Donald Burleson, November 12, 2002
- A value-based bitmap for processing queries on low-cardinality data. (Recommended for up to 1,000 distinct values …
  - Introduction to Adaptive Server IQ, Ch 5, Sybase

Curse of Cardinality: Empirical Evidences



- Index sides, adapted from a presentation by Hakan Jakobsson, ORACLE, 1997 (Stanford Database Seminar)
- 1 million rows (bitmap index compressed with BBC)
- Sizes of compressed bitmap indexes increase with column cardinality – this is generally the case, not just in ORACLE

Curse of Cardinality: Theoretical Evidences

- Analysis of total index size based on Gray Code Ordering (optimal) by Apaydin, Tosun and Ferhatosmanoglu, SSDBM 2008
- Number of columns: A; cardinality of column i: C_i
- Notice the multiplications of column cardinalities of columns in the dataset curse of cardinality

$$E(C_1) + \sum_{i=2}^{A} \left(E(C_i) \prod_{j=1}^{i-1} C_j - C_i \left[\left(\prod_{j=1}^{i-1} C_j \right) - 1 \right] \right)$$

Outline

1. Achilles' Heel of Bitmap Index
2. Order-preserving Bin-based Clustering
3. Analysis
4. Experiment

Ways to Improve Performance of Bitmap Indexes

- Compression
  - Byte-aligned Bitmap Code (BBC), used in ORACLE
  - Word-Aligned Hybrid (WAH) code, used in FastBit, produce optimal bitmap indexes [Wu, et al. TODS 2006]
  - In the worst cases, the index sizes are still larger than B-trees
- Encoding
  - Many bitmap encoding schemes exist, the most compact is the binary encoding
  - The binary encoded index (bit-slice index) is slower than the projection index in the worst case
- Binning
  - Designed to handle high-cardinality data, but needs to scan raw data, which makes it slower than the projection index
  - Solution: Order-preserving Bin-based Clustering (OrBiC)

A Digression: Projection Index

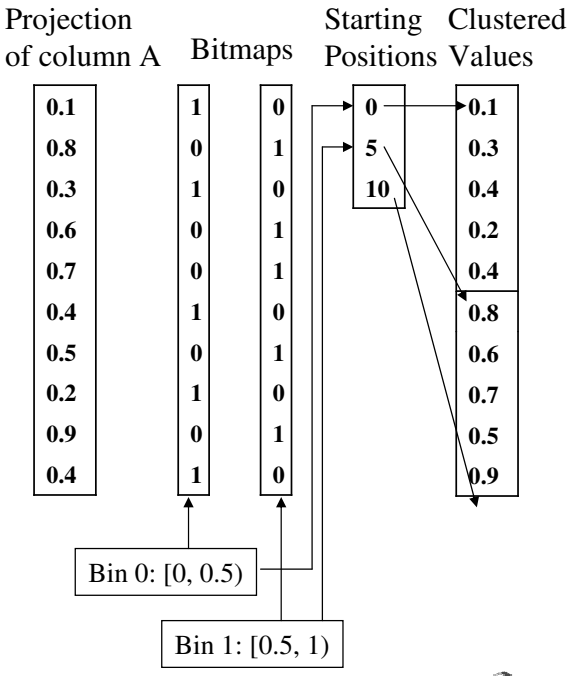
- A projection index is a projection of a column of data [O'Neil and Quass, 1997], also known as the materialized view
- It answers queries by examining **N** values of the column, faster than using B-Tree and other indexes in many cases
- Simplest indexing data structure possible
- Good yardstick to measure any indexing structure

Answering Queries with Binned Index

- Column C (values between 0 and 1)
- Two bins [0, 0.5)[0.5,1), have a bitmap B_0 to represent all rows with 0 <= C < 0.5, and another B_1 for 0.5 <= C < 0.5
- To answer a query involving the condition "C < 0.7", all rows in B_0
- Rows in B_1 are candidates, have to examine the actual values to decide which row satisfy "C < 0.7" – candidate check
- Rows in B_1 are scattered in all pages containing the projection of C
- Candidate check is as expensive as using the projection index to answer the query condition
- To reduce the cost of candidate check, cluster the values according to bins, i.e., OrBiC

OrBiC Data Structure

- OrBiC data structure is an addition to a binned bitmap index
- Let A denote the column name
- With the binned bitmap index shown, all rows in Bin 0 satisfies the query condition "A < 0.7", but rows in Bin 1 are only candidates
- Bin 1 is known as the boundary bin
- Without OrBiC, checking candidates needs to access the base data or a projection of A
  - Usually reads all pages
  - As least as costly as using the projection index
- OrBiC clusters the values needed for candidate check together
  - Reduce the I/O cost



Additional Optimization: Single-Valued Bins

- If a bin contains only a single value, there is no need to store the corresponding values in OrBiC
- It is clear how to construct single-valued bins for integer columns
- It is easy to construct single-valued bins for floating-point valued columns as well
  - For a bin defined as $b_i \leqq A < b_{i+1}$, $b_{i+1}=b_i+|b_i|\varepsilon$ is the smallest value that is larger than b_i, where ε is the machine epsilon or unit round-off error
- In addition to the arrays shown on the previous slide, our implementation of binned bitmap index also stores the actual minimal and maximal values in each bin

Outline

1. Achilles Heel of Bitmap Index
2. Order-preserving Bin-based Clustering
3. Analysis
4. Experiment

Analysis of Binned Index with OrBiC

- B = number of bins
- C = cardinality of the column indexed, C > B
- N = number of rows in the dataset (number of bits in each bitmap)
- w = number of bits in a word, typically, 32 or 64
- Density of i^{th} bitmap, d_i = fraction of bits that are 1, also fraction of values fall in bin i
- Number of words in bitmap i under WAH compression

$$s_i = \left\lfloor \frac{N}{w-1} \right\rfloor + 2 - \left(\left\lfloor \frac{N}{w-1} \right\rfloor - 1 \right) \left((1-d_i)^{2w-2} + d_i^{2w-2} \right)$$

Maximum number of words

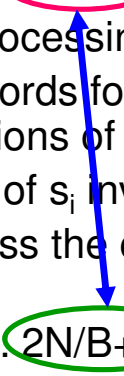
Reduction due to compression

Analysis … Continued

- Size of a binned bitmap index
  - Size of bitmaps, sum of s_i
  - B pointers to the bitmaps, B words
  - B bin boundaries, B words (may use ±1 word depending implementation)
  - B minimal values in each bin, B words
  - B maximum values in each bin, B words
  - Total: 4 B + sum of s_i
- Size of OrBiC data structure
  - B+1 starting positions, B+1 words
  - Cluster values, N words (may be less if there are any single-valued bins)
  - Total: N+B+1

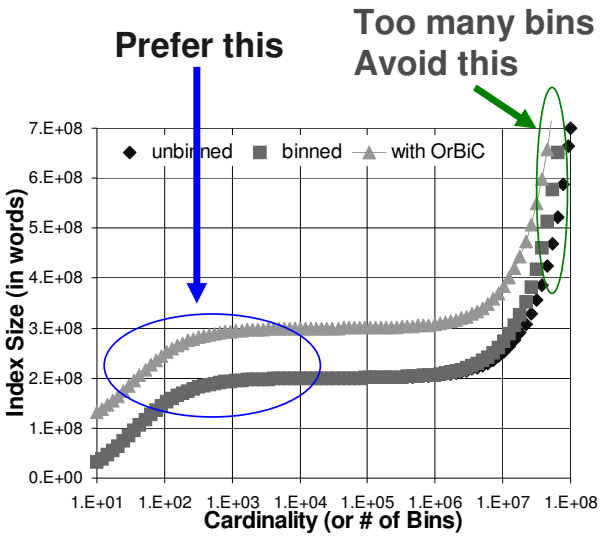
Analysis … Continued

- Query processing cost using a binned bitmap index
  - 4B words for metadata about the index
  - Sum of s_i involved
  - Read N words of the projection of the column for candidate check (may access less words, but often accesses every page containing the projection)
  - Total: N + 4B + sum of s_i
- Query processing cost with OrBiC data structure
  - 5B words for metadata about the index and starting positions of clustered values
  - Sum of s_i involved
  - Access the clustered values for the boundary bins, max 2N/B
  - Total: 2N/B+5B+sum of s_i



Index Sizes

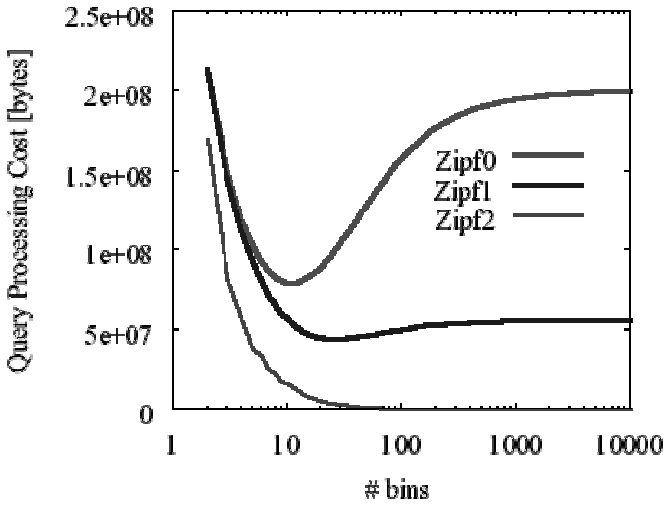
- For random data, WAH compressed index sizes can be given in closed form formulas
  - Zipf data, probability of the ith value proportional to $1/i^z$
  - Uniform random data, z=0
- Using OrBiC with binned indexes increases space requirement
- Choose the number of bins to minimize the query processing costs while keeping the index sizes relatively small
- Minimizing query processing cost must balance two factors
  - Cost due to bitmaps – increases with the number of bins
  - Cost due to candidates – decreases with the number of bins



$N=10^8$

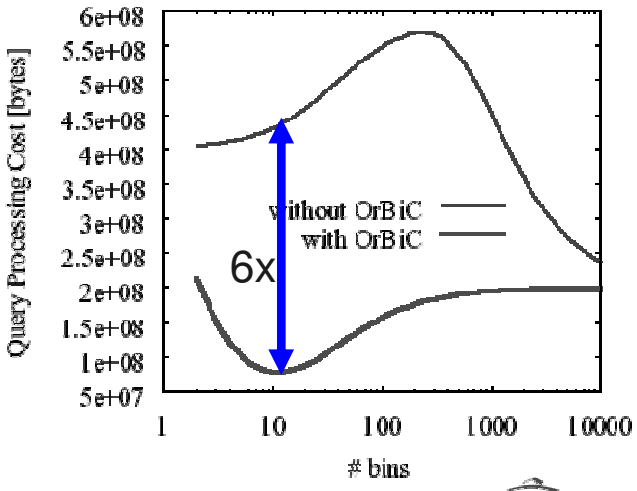
Expected Query Processing Costs on Zipf Data

- The number of bins that minimizes the average query processing costs
- Zipf exponent z = 0: 13, the average cost is about 80MB ($1/5^{th}$ of the projection index, $1/3^{rd}$ of a typical unbinned bitmap index with WAH compression)
- Zipf exponent z = 1: 25
- Zipf exponent z = 2: 550



It Pays to Use OrBiC

- Figure on the right plots the expected average query processing cost against the number of bins for uniform random data
- The query processing costs are always lower with OrBiC than without OrBiC – it is always better to use OrBiC with binned bitmap indexes

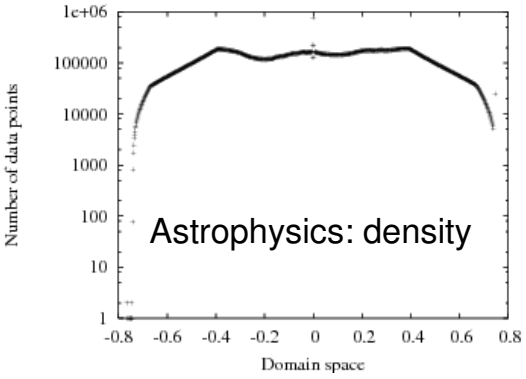


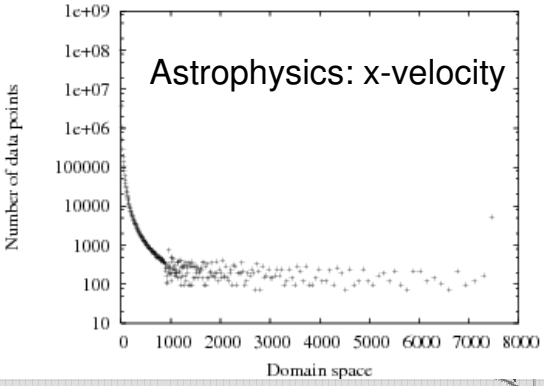
Outline

1. Achilles Heel of Bitmap Index
2. Order-preserving Bin-based Clustering
3. Analysis
4. Experiment

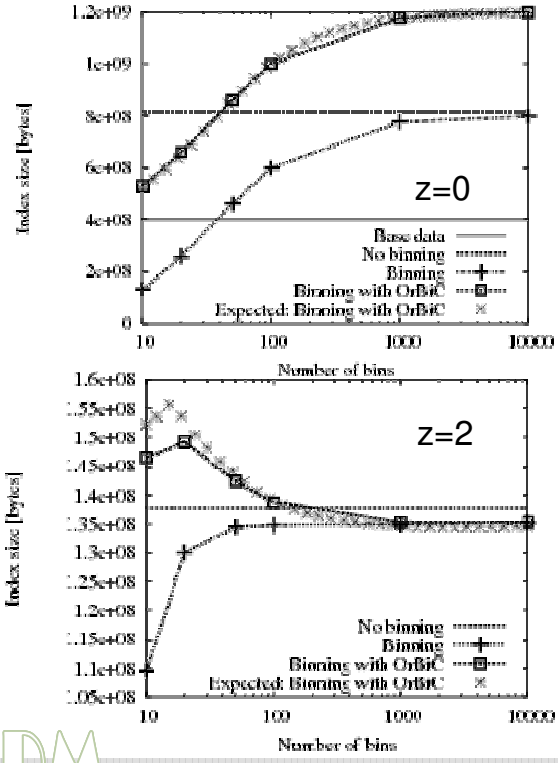
Test Setup

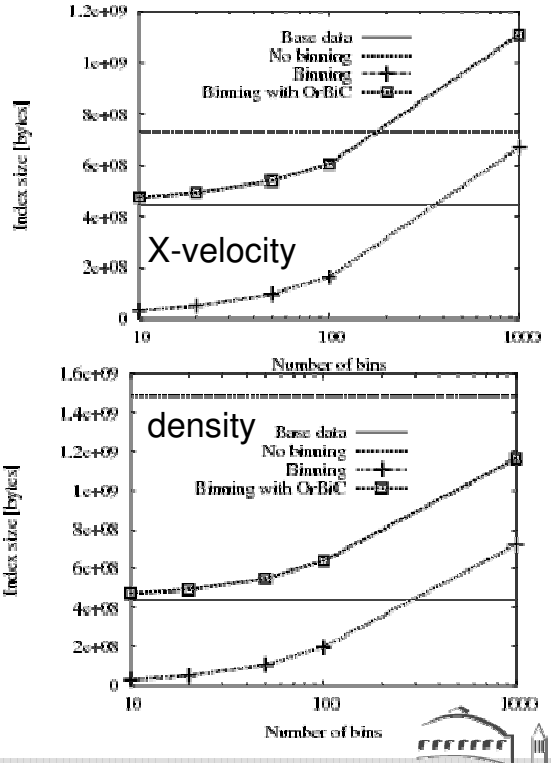
- Two sets of test data are used
  - Synthetic Zipf data: 100 million rows, integer values, cardinality 1 million
  - Astrophysics data: Supernova explosion simulation, 110 million rows, floating-point values, cardinality 20 – 40 million
- Test platform
  - Pentium 4 CPU
  - 2GB RAM
  - RAID-0 with 4 IDE disks (sustainable bandwidth ~60MB/s)
- Test software: FastBit, compiled with GCC 4.1.0
- Software available from http://sdm.lbl.gov/fastbit/



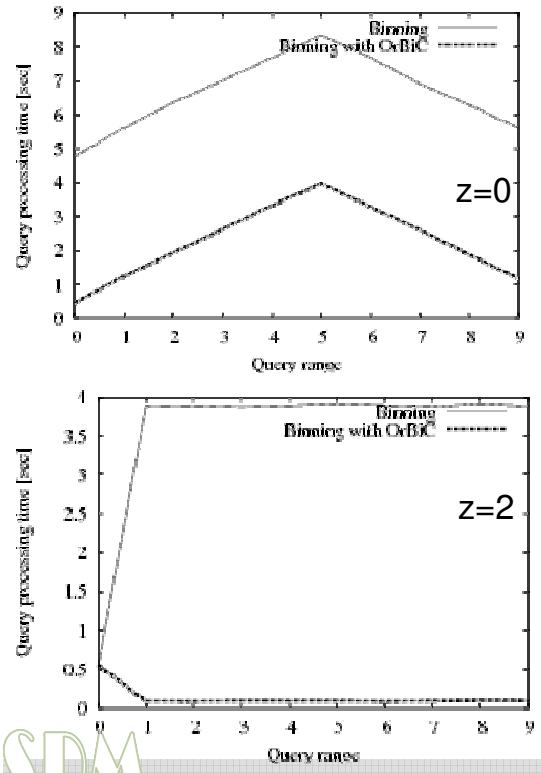


Indexes with OrBiC Have Modest Size as Expected



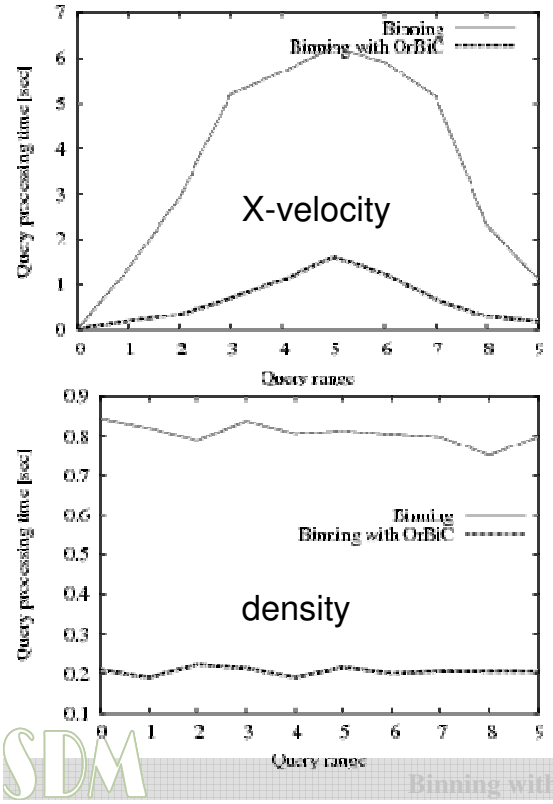


Response Time for Queries on Zipf Data



- On uniform random data, the average speed up expected was about 3
  - The observed speed up is 2.94
  - The observed speed up for Zipf data with z=1 is 5.50, z=2 25.62
- This confirms the theoretical analyses – it pays to use OrBiC
- Speedup on skewed data (z>0) is larger than on uniform random data

Response Time for Queries on Astrophysics Data

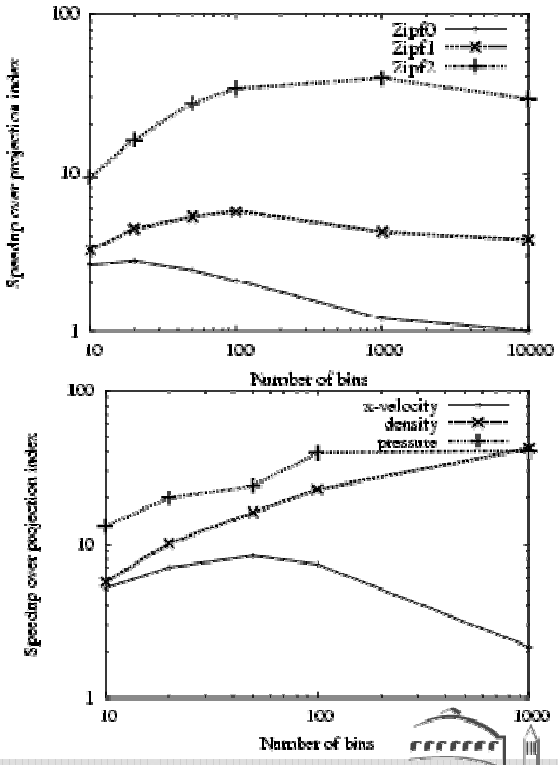


- On real application data, the speedup of using OrBiC is larger than on random data

Column	Speedup	Column	Speedup
Density	3.91	X-velocity	5.65
Entropy	12.61	Y-velocity	4.82
Pressure	4.40	Z-velocity	4.28

Speedup over Project Indexes

- On Zipf data
  - Z=0: max speedup = 3
  - Z=1: max speedup = 6
  - Z=2: max speedup = 26
- On astrophysics data
  - [x|y|z]-velocity: max speedup = 8
  - Density, entropy, pressure: max speedup = 40



Summary

- Order-preserving Bin-based Clustering (OrBiC) enhances binned bitmap indexes by reducing I/O cost
- The effectiveness of OrBiC data structure has been analyzed in theory and demonstrated in timing measurements
- Conclusion: Binning with OrBiC effectively break the curse of cardinality

- Software implementation available under Lesser GNU Public License (LGPL) from http://sdm.lbl.gov/fastbit/

Thanks!

Questions?

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