

Accelerating Data Analytics



Gheorghe Guzun



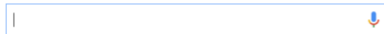
About

- Current position: Assistant Professor of Computer Engineering @ San Jose State University
- PhD @ University of Iowa
- **Research:** data analysis, large-scale indexing, machine learning algorithms, data management systems.
- Email: gheorghii.guzun@sjsu.edu



Big Data Analytics

Google



Google Search

I'm Feeling Lucky



NETFLIX

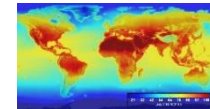
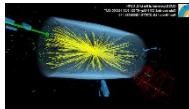


EVERY DAY WE CREATE
**2,500,000,
 000,000,
 000,000**
 (2.5 QUINTILLION) BYTES OF DATA

*This would fill 10 million blu-ray discs,
 the height of which stacked, would measure
 the height of 4 Eiffel Towers on top of one another.*



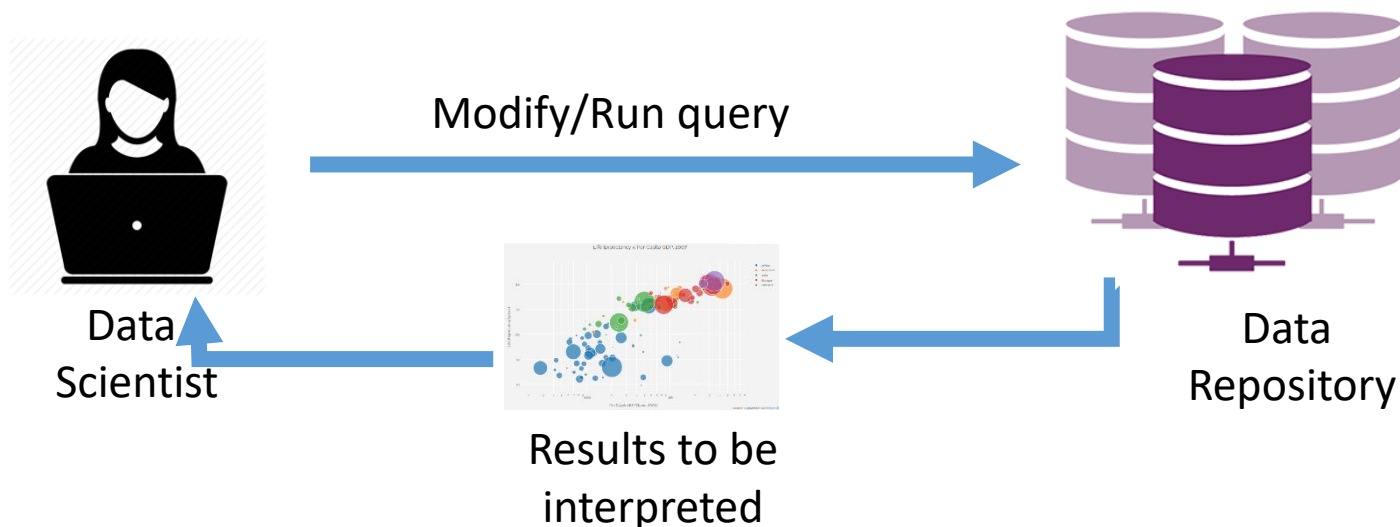
Big Data Analytics



Existing solutions for RDBM Systems are good but challenging to adapt them to the big data architecture which is a share nothing architecture.



Big Data Analytics Challenges



- Data scientists often use exploratory search to extract insight from data
- Efficiency of existing data analytics tools degrades with increasing data volumes
- Bottlenecks: I/O, Memory, Communication costs, CPU

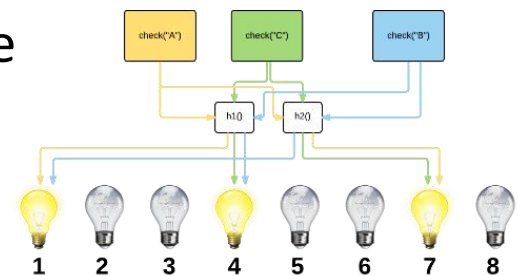
Outline

- Introduction
- Background on bit-vector indexing
- Extending bit-vector indexing for big data analytics
 - Answering complex queries
 - Partitioning and distributed query algorithms
- Scalable Bit-vector compression
 - Hybrid Compression
 - VAL - Variable Aligned Length compression
- Other projects



Bit-Vectors for Big Data

- Bit vectors widely used in existing data warehouses
 - Oracle, Sybase, Vertica, MongoDB, Snowflake
- Advantages of bit-vectors:
 - Simple but powerful
 - Leverage fast bit-wise operations to resolve queries-supported by hardware
 - Compact data representation
 - Highly compressible (hybrid run-length encoding)
 - Vector processing and SIMD operations
 - Efficient partial result combination and aggregation



Bit-Vector Indexing

- An array of bits
 - Each bit in the array corresponds to an object in the data table
 - Encodes a Boolean state of the respective object

ID	Value
0	2
1	3
2	0
3	1

Bitmap ≥ 2
1
1
0
0

Range encoded
bitmap index

Bit-slice	
B1(2^1)	B0(2^0)
1	0
1	1
0	0
0	1

Bit-sliced
index



SUM Aggregation with Bit-Sliced Index

Store sales
5
13
2
6
7

Bit-Sliced Index

B3: 01000

B2: 11011

B1: 00111

B0: 11001

Count of B4: 1

Count of B3: 4

Count of B2: 3

Count of B1: 3

$$5 + 13 + 2 + 6 + 7 \\ = 33$$

$$1 * 2^3 + 4 * 2^2 + 3 * 2^1 + 3 * 2^0 \\ = 8 + 16 + 6 + 3 \\ = 33$$



SUM through Bitwise Operations

Value	Value
2	0
3	2
0	1
1	3

+

=

Value
2
5
1
4

Value	Value
00000000...10	00000000...10
00000000...11	00000000...11
00000000...00	00000000...00
00000000...01	00000000...01

+

Bit-slice	Bit-slice
2^1	2^0
1	0
1	0
0	1
0	1

Uses less bits per value, and only AND, XOR and OR bitwise operations

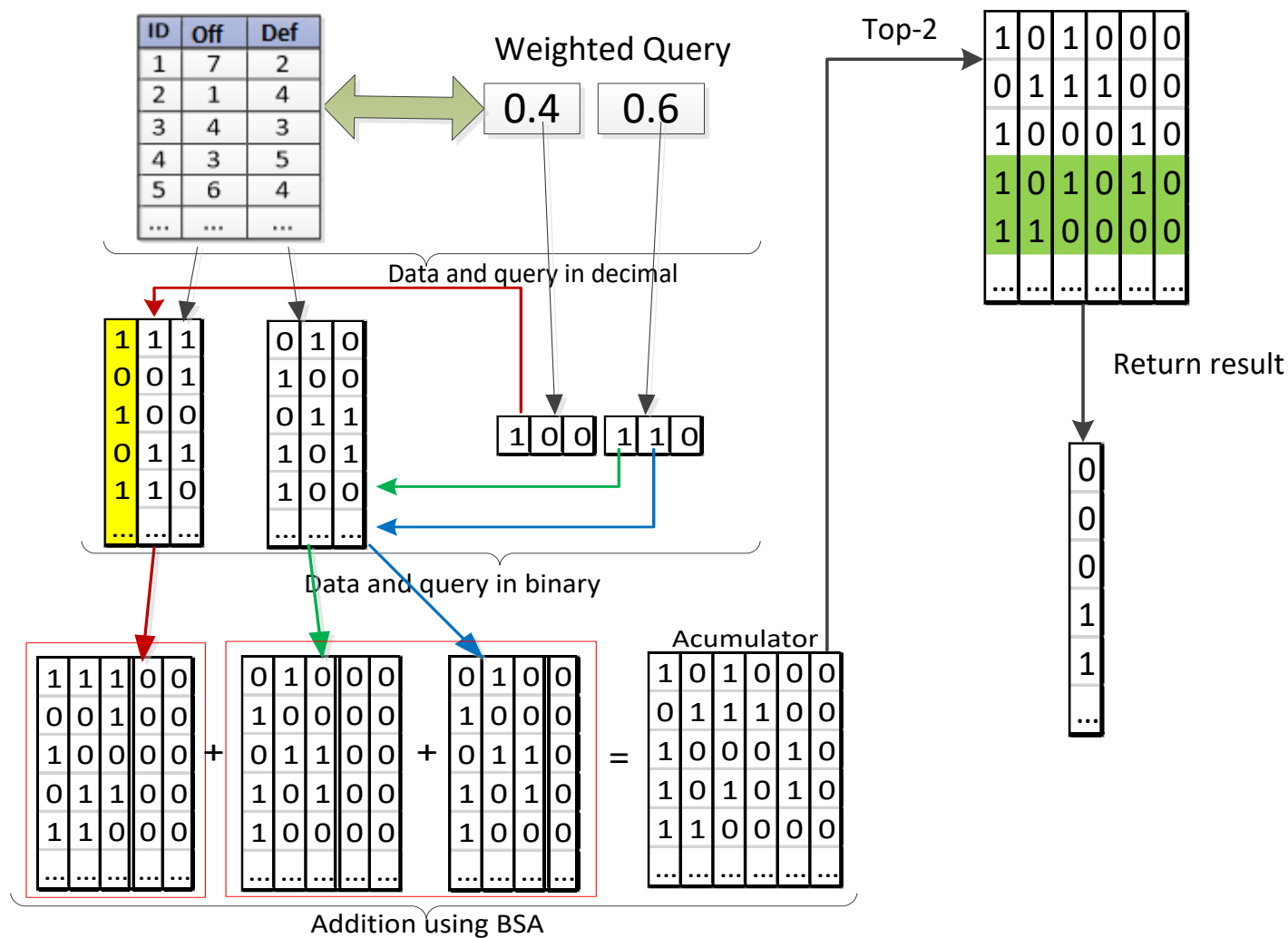
Complex Query Example

- Top-K query processing
 - Apply the ranking function F to all objects
 - Sort weighted objects based on their score
 - Return k best objects
- Requires aggregation across all dimensions
 - Not suitable for hash accesses
- Sorted lists - not suitable for high dimensions
 - High dimensionality -> large list of candidate objects to maintain
- How can we solve this efficiently in a distributed environment?

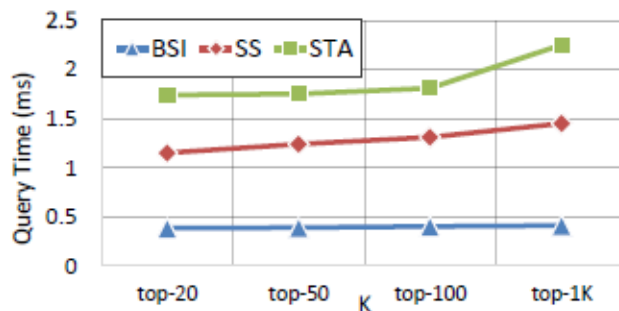
NBA Player / Year	GP	PTS	REB	AST
Wilt,Chamberlain / 1967	82	1992	1952	702
Billy,Cunningham / 1972	84	2028	1012	530
Kevin,Garnett / 2002	82	1883	1102	495
Julius,Erving / 1974	84	2343	914	462
Kareem,Abdul-jabbar / 1975	82	2275	1383	413



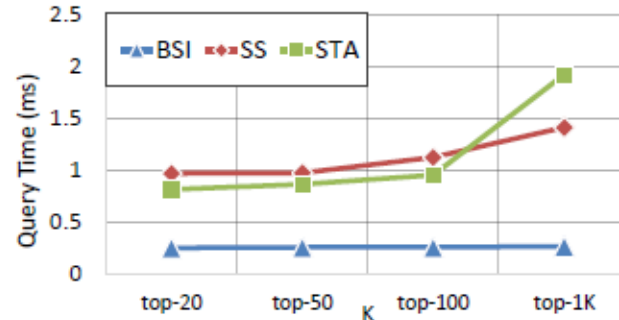
Top-K Query Using BSI



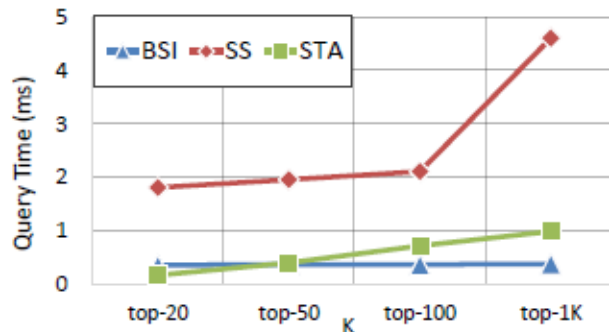
BSI vs Other Approaches for top-k Preference Queries



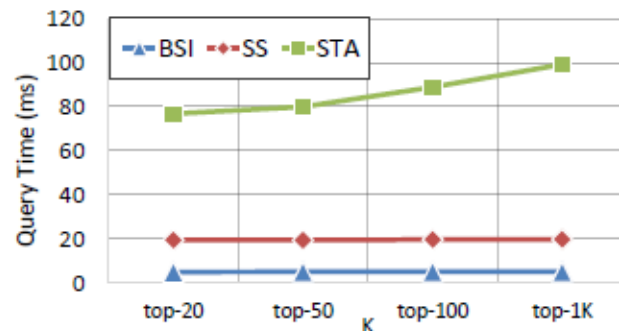
(a) Top-k for the coil2000 data set.
Rows: 9822, Attributes: 86



(b) Top-k for the internet data set.
Rows: 10104, Attributes: 72



(c) Top-k for the kegg-metabolic data set.
Rows: 53413, Attributes: 24

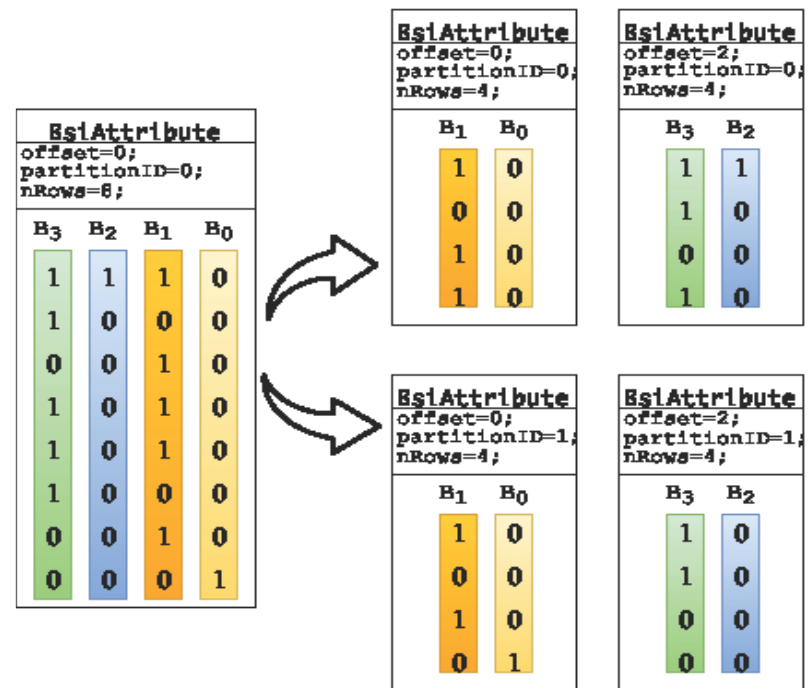


(d) Top-k for the poker-hand data set.
Rows: 1,000,000; Attributes: 11

Efficient for centralized systems. How do we extend it for distributed environments?

Partitioning of the BSI Attribute

- Number of partitions determined by available hardware resources
- Challenges:
 - Metadata needed to ensure correctness.
 - Tradeoff between parallelism and communication cost
 - Load balancing



Complex queries on distributed BSI

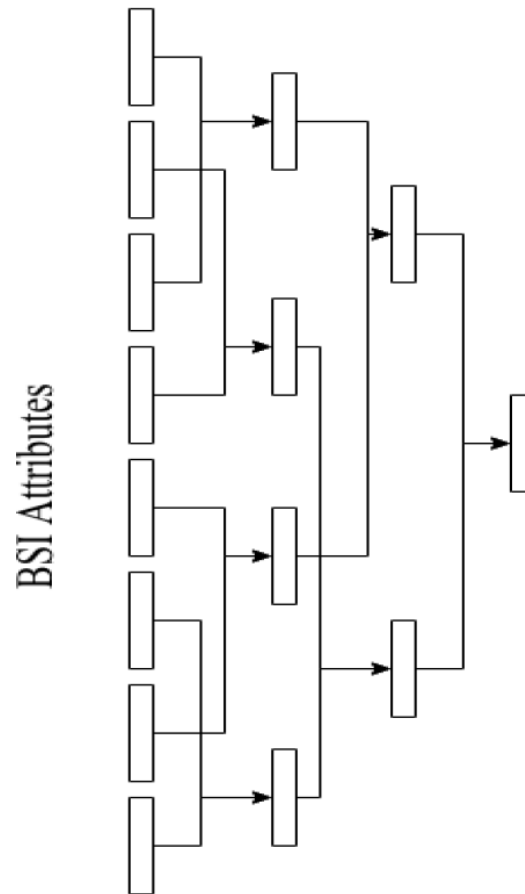
- Top-K preference query

- Use parallel MULTIPLY BY CONSTANT to apply query weights
- Perform parallel SUM and aggregate the results
- Use BSI top-K Max to extract the top K results

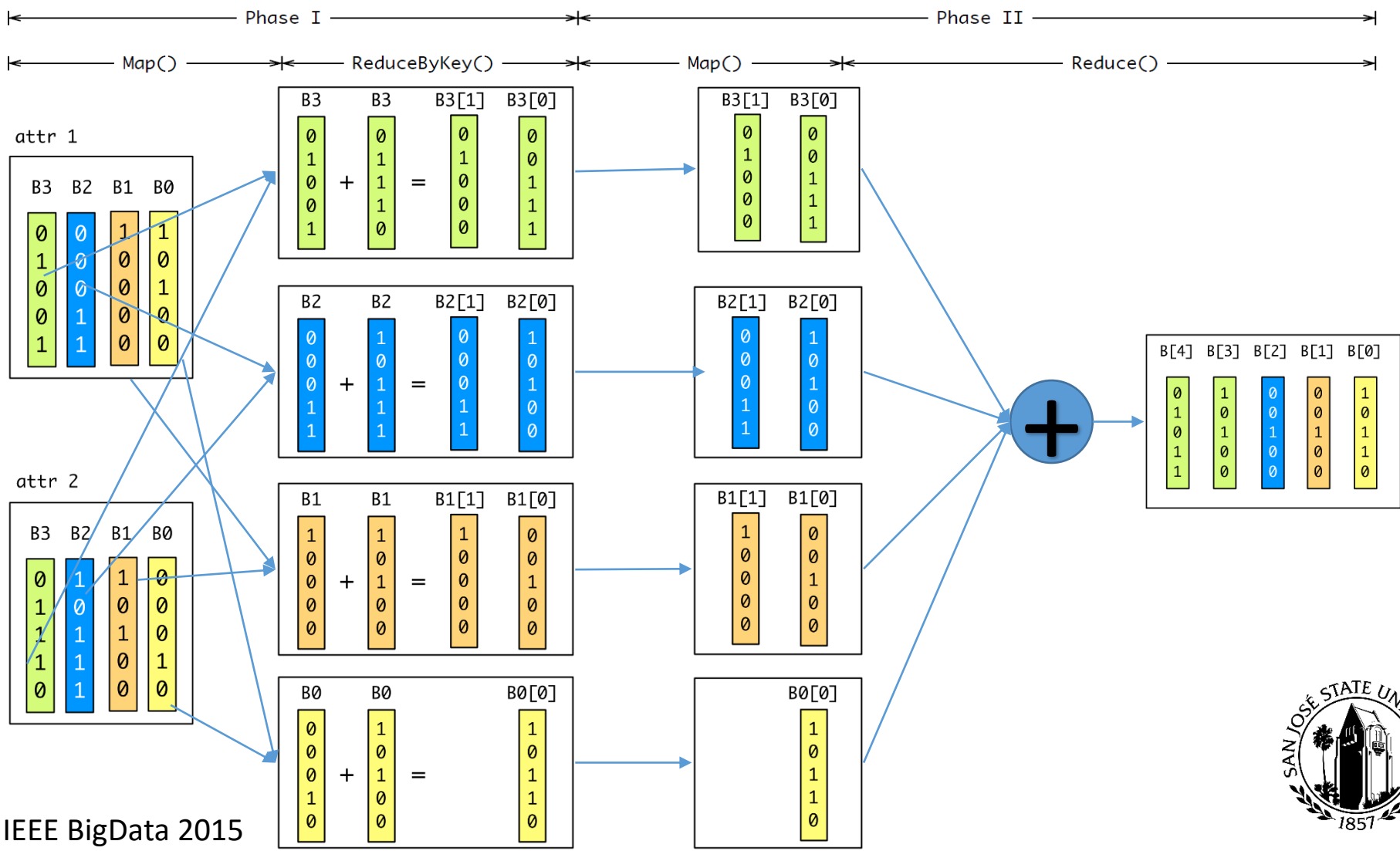


BSI Attribute Aggregation with Tree Reduction

- Tree reduction with BSI:
Baseline map-reduce
 - High number of reduction rounds
 - High data shuffling
 - Poor load balancing



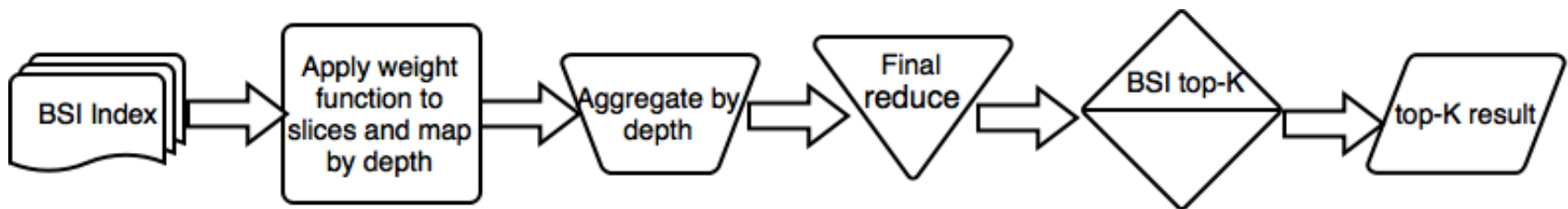
Distributed Aggregation using BSI Slice Mapping



Complex queries on distributed BSI

- Top-K preference query

- Use parallel MULTIPLY BY CONSTANT to apply query weights
- Use parallel SUM to add slices by depth
- Use parallel SUM to aggregate the final sum
- Use BSI top-K Max to extract the top K results



Top-K Performance Results for Distributed BSI

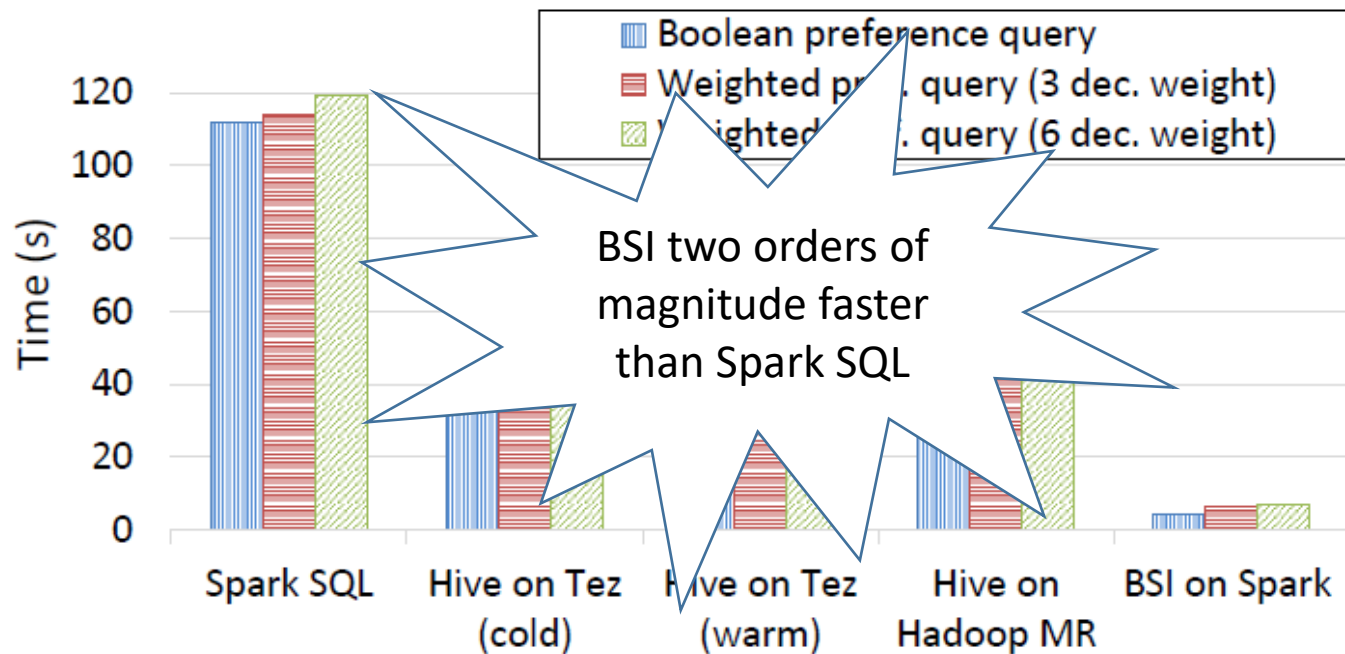


Figure 12: BSI top-K preference and top-K weighted preference query time compared to Hive on Hadoop map-reduce, and Hive on Tez (Dataset: HIGGS, 32 bit-slices per dimension).



Outline

- Introduction
- Background on bit-vector indexing
- Extending bit-vector indexing for big data analytics
 - Answering complex queries
 - Partitioning and distributed query algorithms
- Scalable Bit-vector compression
 - Hybrid Compression
 - VAL - Variable Aligned Length compression
- Other projects



Bit-Vector Compression

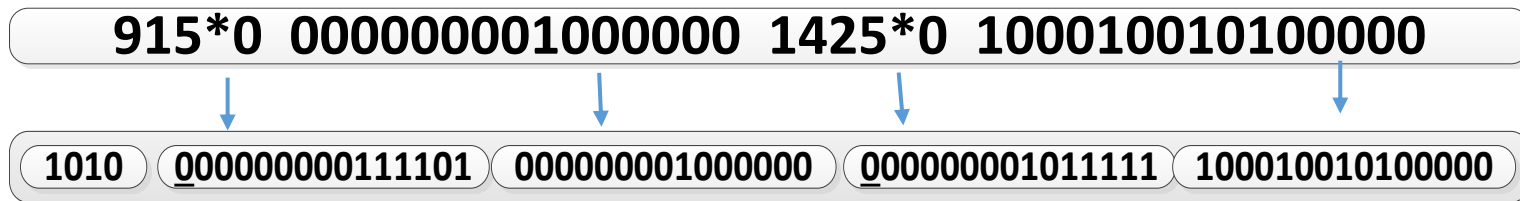
- Goal:
 - Reduce bit-vector size
 - Minimize overhead in query execution time
 - Execute query without explicit decompression
 - Run-length encoders
 - Word-Aligned Hybrid Code (WAH) – widely adopted
 - Two types of words - simple
 - Word Aligned \rightarrow Faster CPU time
 - EWAH, PLWAH, CONCISE, COMPAX are word-aligned based

133 Bits	1,20*0,4*1,78*0,30*1
31-bit groups	1,20*0,4*1,6*0 62*0 10*0,21*1 9*1
groups in hex	400003C0 00000000 00000000 001FFFFFF 000001FF
WAH (hex)	400003C0 80000002 001FFFFFF 000001FF

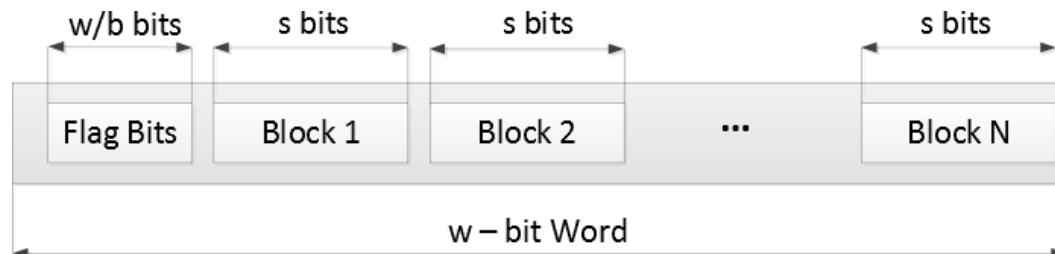
Literal Word Fill Word



VAL Compression



- Compresses given segment length
- Operates directly on compressed data
- Longer segment lengths good for long runs

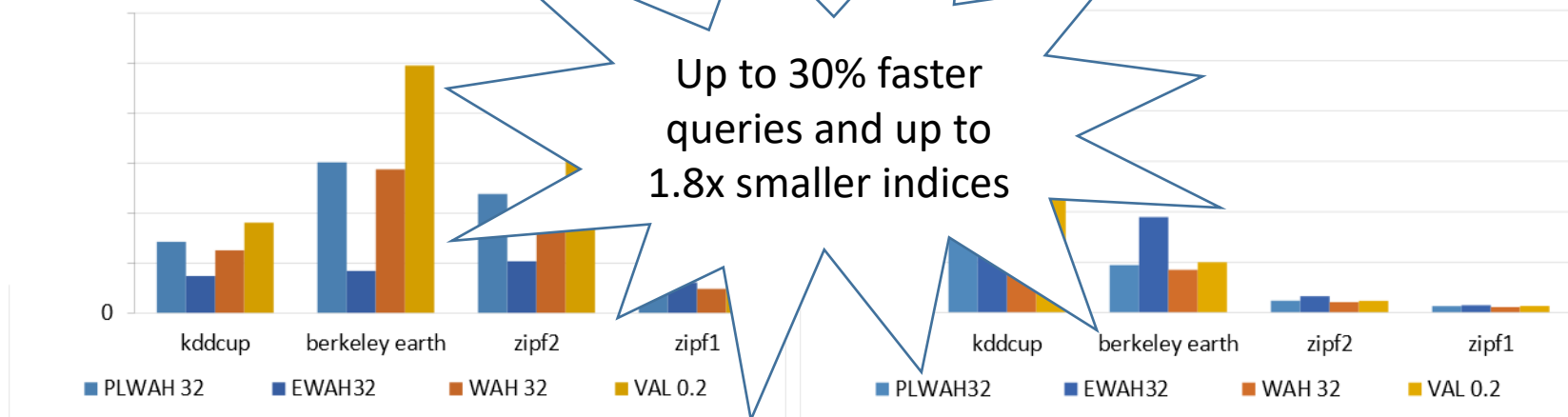


VAL-WAH: Combined gain

- To help simplify discussion on trade-off between compression and query time:

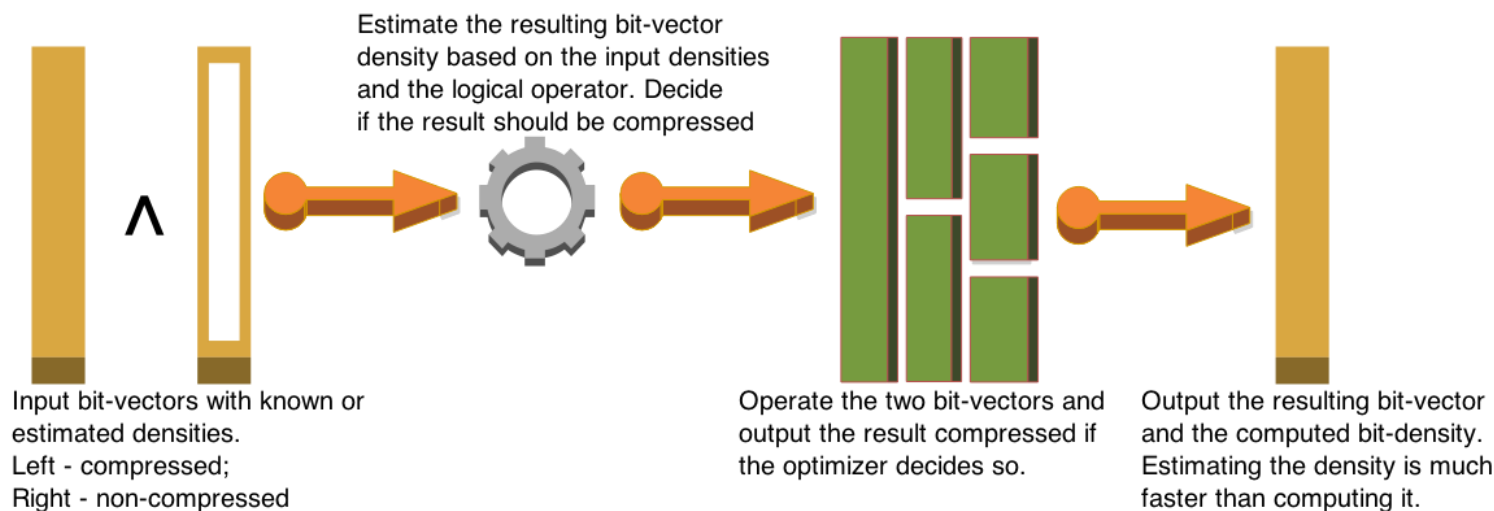
$$\text{gain} = \frac{1}{H_m} = \frac{\text{queryTimeRatio} + \text{compRatio}}{2 \times \text{queryTimeRatio} \times \text{compRatio}}$$

Up to 30% faster queries and up to 1.8x smaller indices

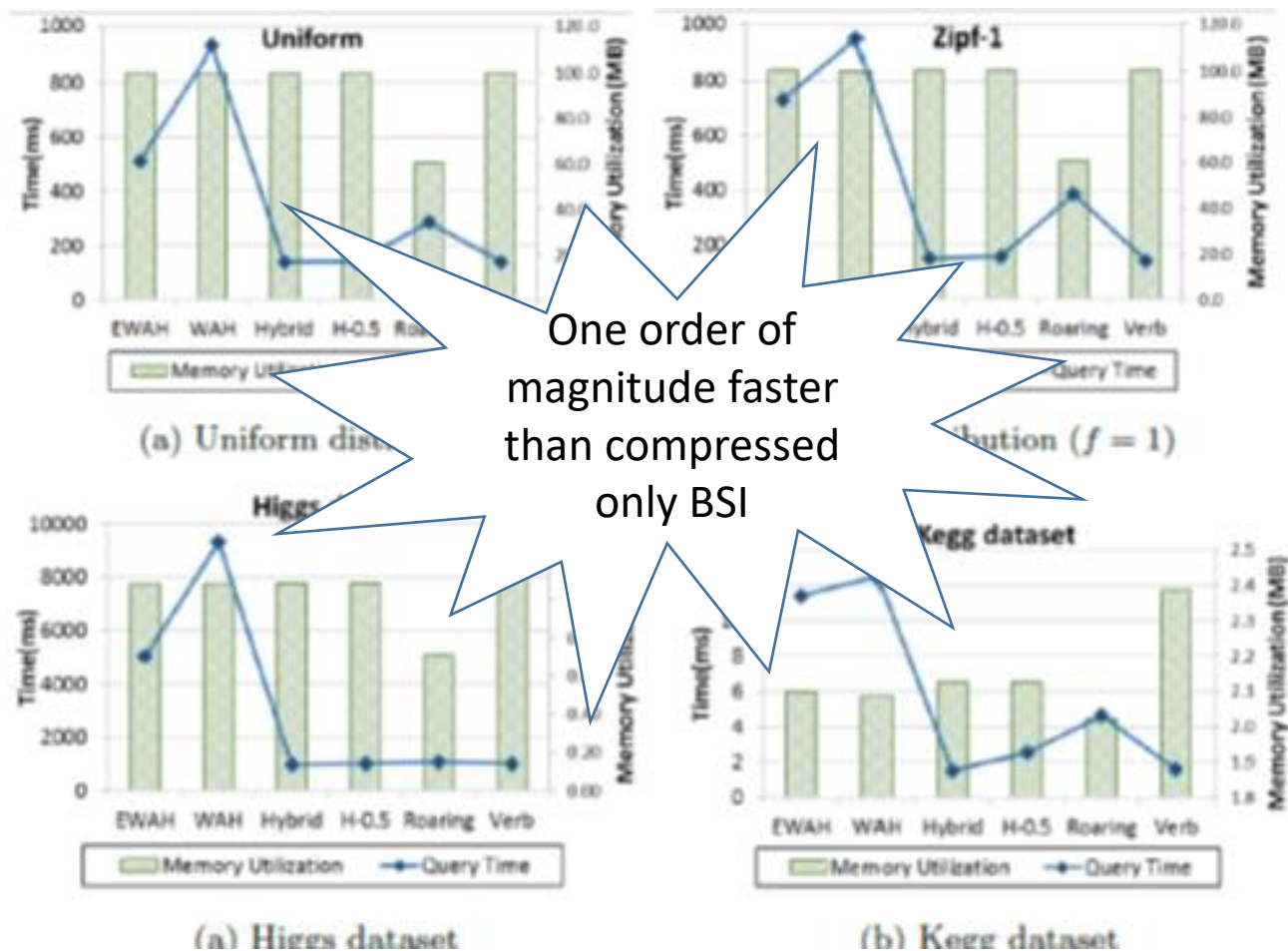


Hybrid Compression for Dense Bit-Vectors

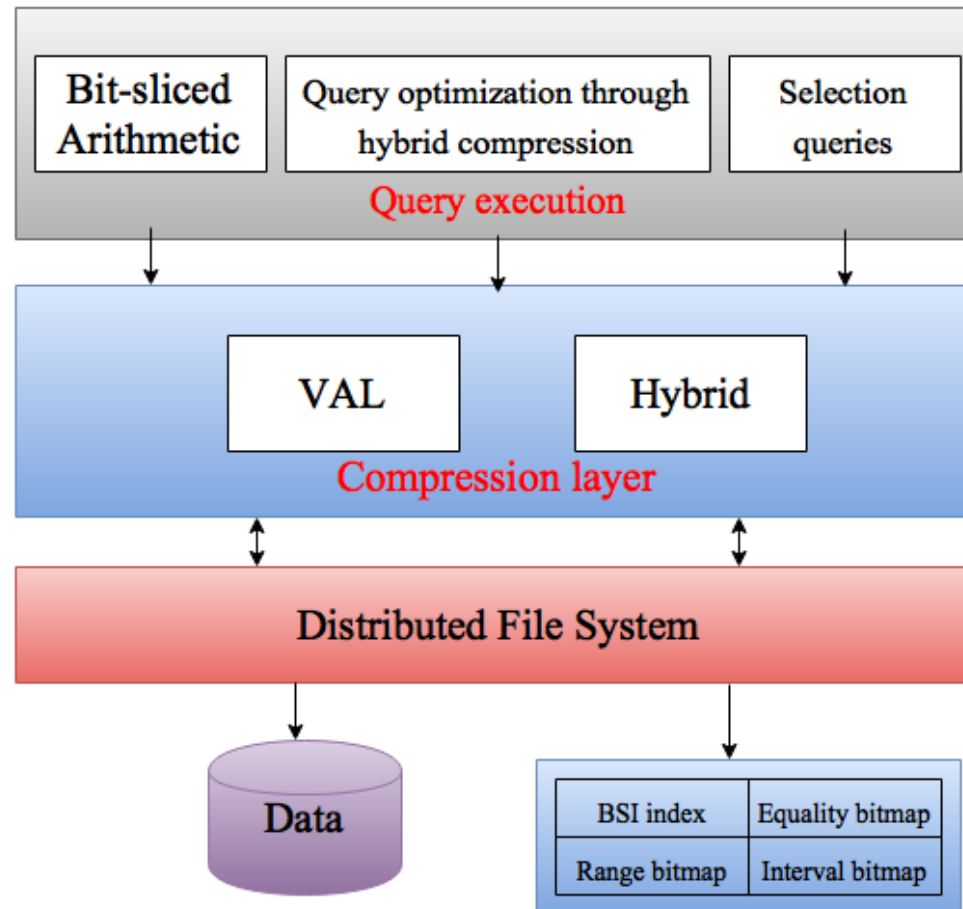
- A combination of compressed and the verbatim scheme
 - There is a tradeoff when compressing
 - BSI starts with dense vectors
- Optimized for speeding-up query execution



Hybrid Compression performance results



Indexing and Query System Stack



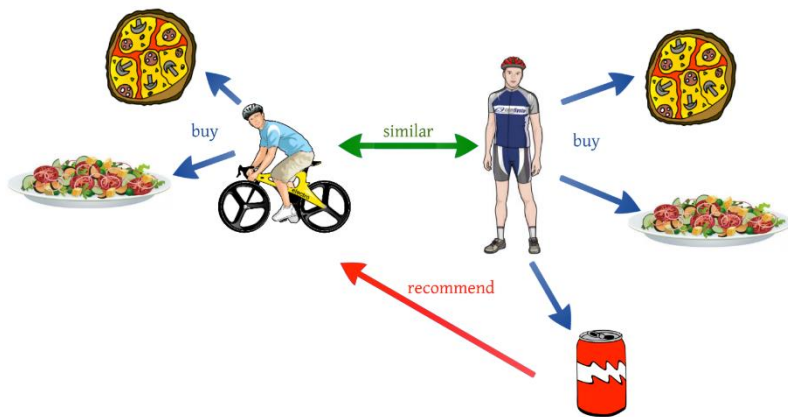
Outline

- Introduction
- Background on bit-vector indexing
- Extending bit-vector indexing for big data analytics
 - Answering complex queries
 - Partitioning and distributed query algorithms
- Scalable Bit-vector compression
 - Hybrid Compression
 - VAL - Variable Aligned Length compression
- Other projects

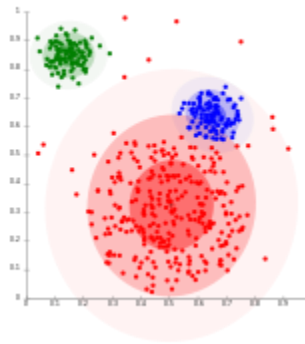


Other projects

- Efficient similarity search using bit-vectors
- Identify more applications and domains that can benefit from distributed indexing and query processing



Recommender systems



Data clustering

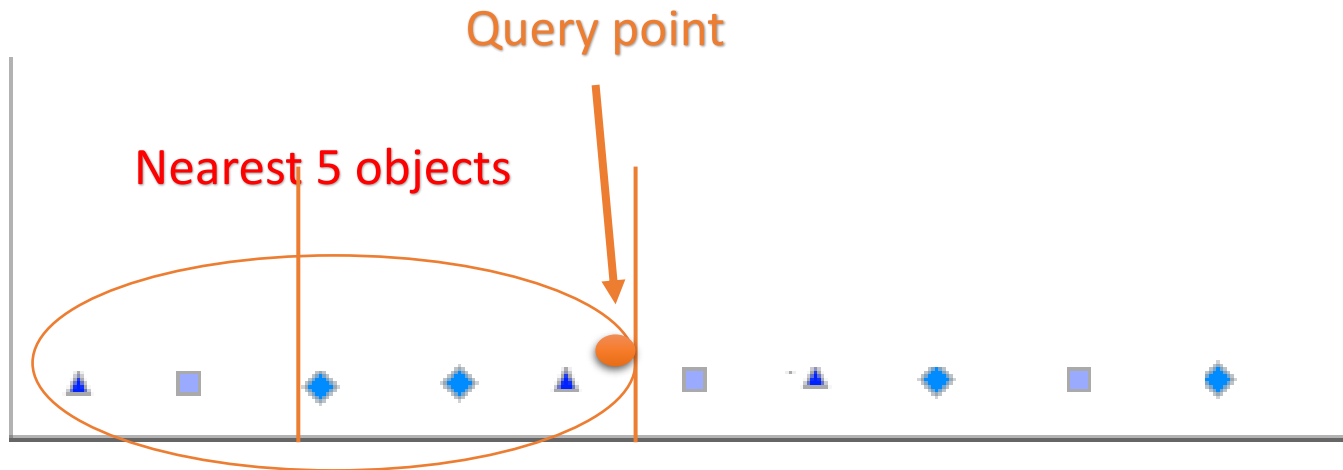


Data privacy and security

Other projects

- More meaningful similarity in high dimensional data

Query Aware Quantization



Current on-going work

- Scalable Indexing for various types of data
- Explore more queries
- Explore new data structures for distributed indexing.
- Trade-off accuracy for performance
 - Approximate results are OK for some applications
 - Look at the most significant data only to improve performance
- Specialized hardware for indexing support
 - Accelerate further: Enable indexing streaming data



Thank You!

Questions?

Collaborators:

Guadalupe
Canahuate,
David Chiu,
Joel Tosado,
Jason Sawin,
Ricardo Mantilla,
Josiah McClurg,
Raghuraman
Mudumbai,
Hyeran Jeon



References

- Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010
- Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.
- Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.
- Guzun, Gheorghi and Canahuate, Guadalupe and Chiu, Dereck and Sawin, Jason. A tunable compression framework for bitmap indices. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 484–495. IEEE, 2014.
- Guzun, Gheorghi and Canahuate, Guadalupe. Performance evaluation of word-aligned compression methods for bitmap indices. *Knowledge and Information Systems*, pages 1–28, 2015.
- Guzun, Gheorghi and Tosado, Joel and Canahuate, Guadalupe. Slicing the Dimensionality: Top-k Query Processing for High-Dimensional Spaces. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XIV*, pages 26–50. Springer, 2014.
- Guzun, Gheorghi, and Guadalupe Canahuate. "Hybrid query optimization for hard-to-compress bit-vectors." *The VLDB Journal* (2015): 1-16.
- Guzun, Gheorghi, Joel E. Tosado, and Guadalupe Canahuate. "Scalable preference queries for high-dimensional data using map-reduce." *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015.

