Integrating Compression and Execution in Column-Oriented Database Systems

paper 2006

张朝威
2017.01.12

大纲

- Purpose 目的

- explore the performance and architectural implications of integrating a wide range of compression schemes into a column-oriented database. 探讨将压缩体系整合到列存数据库系统带来的性能和构架方面的影响

- Focus 焦点

- using compression to maximize query performance, not to minimize storage sizes 使用压缩来最大化查询性能，而非最小化存储空间

- Contents 主要内容

- 介绍常规压缩算法，以及它们在列存数据库中的使用

- 通过实验来说明压缩算法、数据特征和查询负载之间的平衡，并提出决策树来辅助数据库设计

- 介绍一个允许直接在压缩数据上进行操作的查询器构架

- illustrate the importance of introducing as much order as possible into columns and demonstrate the value of having secondary and tertiary sort orders 多列排序的重要性

大纲

- 相关工作
- C-STORE 构架
- 压缩
- 基于压缩的查询
- 实验结果
- 结论

相关工作

- 90 年前后，压缩的研究：减少存储空间 -> 对 DB 的性能影响；
- 前提条件：压缩 (解压)cost >> IO cost,

  reduce CPU cost　尽量减少 CPU 代价
- 1. light-weight compression(sub-optimal) 使用轻量级压缩算法
- 2. lazy decompression 延迟解压
- 3. integrating knowledge about compression into the query executor and allowing some amount of operation directly on compressed data 让执行器知道压缩的相关知识，允许对压缩数据直接进行操作

相关工作

Our work focus on 2 points ：
1. column-oriented compression algorithms 面向列存的压缩算法
2. direct operation on compressed data 直接在压缩数据上操作

	Compression/Decompression	Operation on compressed data
Previous Work	when operations cannot simply compare compressed values, they must be performed on decompressed tuples 如果不能够在压缩数值上进行简单比较，就需要整个解压才能够操作	Decompressing the data before it reaches the operators unless dictionary compression is used and the data can be processed directly 字典压缩才能够直接在压缩数值上运算；否得就得直接解压
Our Work	column-oriented compression schemes provide further opportunity for direct operation on compressed data 从压缩算法来探索更多可以直接在压缩数值上进行的操作	introduces a novel architecture for passing compressed data between operators that minimizes operator code complexity while maximizing opportunities for direct operation on compressed data 在操作符之间传递压缩数值

C-STORE 构架

- Each table is physically represented as a collection of projections 表用 projection 集合来表示
- Each projection consists of a set of columns along with a common sort order for those columns. 一个 projection 由一组列组成，同时可指定多个排序列

  (shipdate, quantity, retflag, suppkey | *shipdate, quantity, retflag)*
- store the same column in multiple projections, each with a different sort order 同一个列可能会被存储多份
- join indices ，用来从不同 projection 中抽取数据得到完整的元组

C-STORE 构架

4 种主要的操作

- SELECTION operators produce bit-columns that can be efficiently combined 选择操作会使用到 bit-columns
- A special PERMUTE operator is used to reorder a column using a join index 重排序
- PROJECTION is free since it requires no changes to the data, and two projections in the same order can be concatenated for free as well. 投影操作则很简单
- JOINs produce positions rather than values 连接产生的是位置而非数值

参考《C-Store: A Column-oriented DBMS》

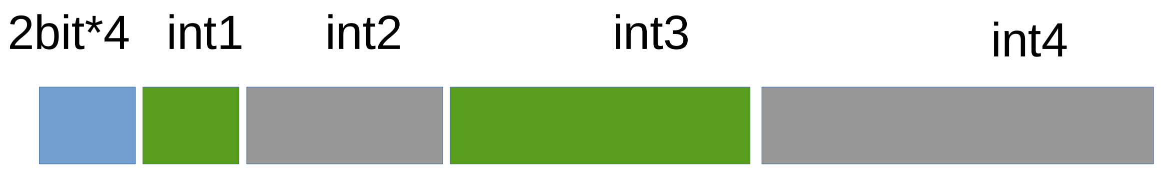
压缩

- NULL Suppression
- Dictionary Encoding
- Run-Length Encoding
- Bit-Vector Encoding
- Lempel-Ziv Encoding

压缩 - NULL Suppression

- Idea: consecutive zeros in the data are deleted and replaced with a description of how many there were and where they existed. 剔除掉连续的 0 值，并使用简要的描述信息来替换；这依赖于数值出现的频率
- 面向列存的压缩方法

- 1. 主要参考《 The Implementation and Performance of Compressed Databases 》论文。如果采用字典压缩方法，可以使用一个 entry code 来表示 NULL; 如果整数或者浮点数，需要特殊处理并可以区别数值 0 ； 如果为字符串类型，需要能够区别 empty string
- 2. allow field sizes to be variable and encode the number of bytes needed to store each field in a field prefix 例如， 2bit 来表示对应整数的字节数目； 那么就会把 4 个整数放在一起，令其 2ibt 合为 1B ，后面连续 4 个变长整数
-
-



- 3. keep byte aligned 保留字节对齐

压缩 - Dictionary Encoding

- Key point:replace frequent patterns with smaller codes
- 面向列存的压缩

1. 计算出单列有效数值的个数；
2. 计算出表达这些数值需要的 bit X;
3. 计算出具体数值对应哪个 code;
4. 进行字典 encode;

decided by requiring the dictionary
to fit in the L2 cache
（使用字节对齐）

通过简单的位移就可以在直接在压缩数据上操作
（使用定长编码）

举例：

1. 某个列表示月内日期，有效值为 32（包括 NULL）；
2. 有效的 bit 使用 5 即可表达所有数值；
3. 采用 3-value/2-byte 方案进行存储如下：
   X000000000100010 -> 31 25 1
4. 进行字典 encode ，保持字节对齐；

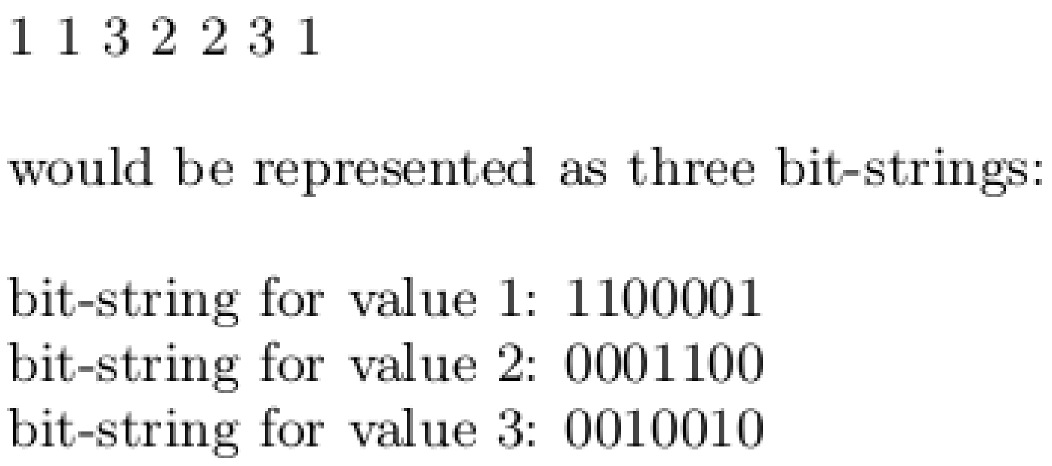
数值	编码
1	00000
25	00001
31	00001

压缩 - Run-Length Encoding

- Key point:compresses runs of the same value in a column to a compact singular representation
- 面向列存的压缩
  1. 表达为定长的三元组（ value, start position, run length ）
  2. 进行排序后压缩效果会好；

压缩 - Bit-Vector Encoding

- Key point: 适用于该列有效值很少的场景，其原理类似于行存表的 bitmap
- 面向列存的压缩



1 1 3 2 2 3 1

would be represented as three bit-strings:

bit-string for value 1: 1100001
bit-string for value 2: 0001100
bit-string for value 3: 0010010

压缩 - Lempel-Ziv Encoding

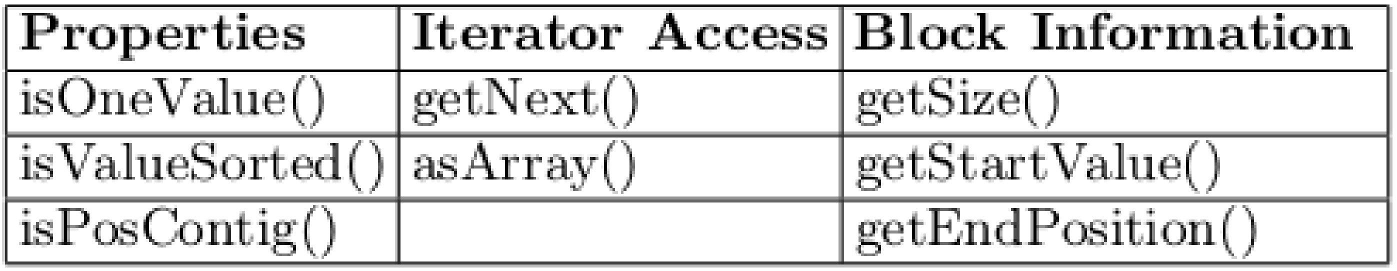
- Key point: takes variable sized patterns and replaces them with fixed length codes 找到变长的模式，将它们替换为定长的编码
- 主要过程 : to parse the input sequence into non-overlapping blocks of different lengths while constructing a dictionary of blocks seen so far. Subsequent appearances of these blocks are replaced by a pointer to an earlier occurrence of the same block. 将输入序列解析为不重叠的、长度不一的块，同时建立字典信息；后续出现这些块时，用已出现块的位置来替换
- 面向列存的压缩

  使用了开源的、优化后的 LZ 压缩算法；

  参考 http://www.lzop.org ；

基于压缩的查询

- Query Executor Architecture
- Compression-Aware Optimizations



Properties	**Iterator Access**	**Block Information**
isOneValue()	getNext()	getSize()
isValueSorted()	asArray()	getStartValue()
isPosContig()		getEndPosition()

Table 1: Compressed Block API

Query Executor Architecture

- 类 Compression block

  它是压缩数据的中间表示，一个 compression block 包含了压缩格式的列数据缓存；

  同时提供了外层访问该数据缓存的 API ，用于 operate directly on compressed data ；

- 如果解压不可避免

  1. getNext()

     1.1 临时解压 next value ；

     1.2 返回该数值和其位置；

     1.3 重复上面 2 步直接所有数值被访问；

  2. asArray()

     2.1 直接将整个缓存都解压；

     2.2 返回该缓存的指针，以及缓存大小；

     2.3 上层循环访问该缓存的所有解压数值；

Operate directly on compressed data

1. getSize()
2. getStartValue()
3. getEndPosition()

上面三个函数对于不同的压缩方法，具体的含义有所区别

	RLE	Bit Vector
getSize()	返回 run length	返回 bit=1 的个数
getStartValue()	返回 value	返回对应 value
getEndPosition()	返回 start pos + run length − 1	返回最后一个 bit=1 的位置

Query Executor Architecture

- 类 DataSource operator

  1. 它是查询引擎和存储引擎之间的接口；

  2. 具有压缩相关知识，知道磁盘页压缩元信息，以及哪些索引适用于该列；

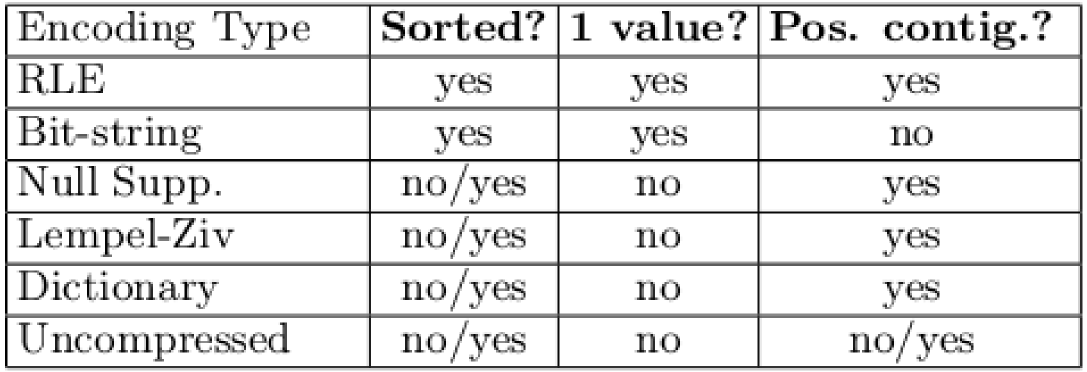
  3. 从磁盘上读压缩页，然后将数据填充到 compression block 中；像 LZ 此类的压缩方式，则直接解压放入其中；

  4. 可以将等值谓词直接下推到该类中进行运算；

Compression-Aware Optimizations

- abstracting away the properties of compressed data that allow the operators to perform optimizations when processing

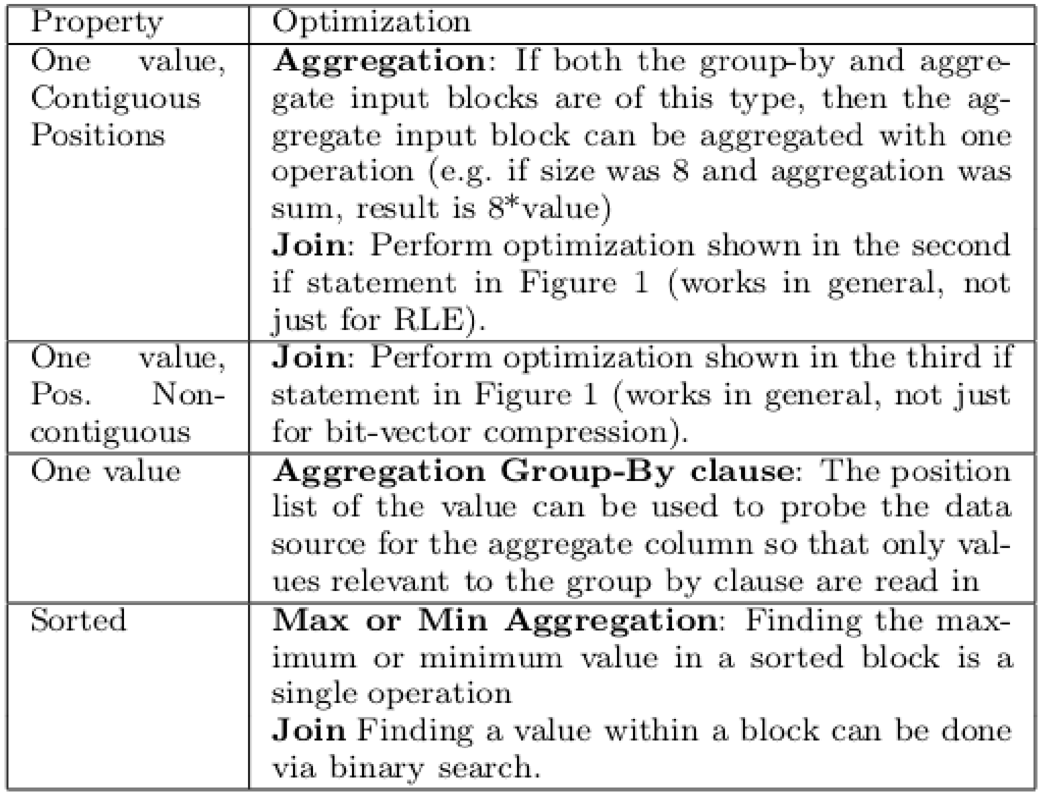
属性	描述
isOneValue()	压缩块只包含仅仅一个数值
isValueSorted()	压缩块中的数值是否有序
isPosContig()	压缩块中的数值是列中的连续的子集



Encoding Type	Sorted?	1 value?	Pos. contig.?
RLE	yes	yes	yes
Bit-string	yes	yes	no
Null Supp.	no/yes	no	yes
Lempel-Ziv	no/yes	no	yes
Dictionary	no/yes	no	yes
Uncompressed	no/yes	no	no/yes

备注： Yes/no 说明此属性信赖于具体的数据而定

Compression-Aware Optimizations



Property	Optimization
One value, Contiguous Positions	**Aggregation**: If both the group-by and aggregate input blocks are of this type, then the aggregate input block can be aggregated with one operation (e.g. if size was 8 and aggregation was sum, result is 8*value) **Join**: Perform optimization shown in the second if statement in Figure 1 (works in general, not just for RLE).
One value, Pos. Non-contiguous	**Join**: Perform optimization shown in the third if statement in Figure 1 (works in general, not just for bit-vector compression).
One value	**Aggregation Group-By clause**: The position list of the value can be used to probe the data source for the aggregate column so that only values relevant to the group by clause are read in
Sorted	**Max or Min Aggregation**: Finding the maximum or minimum value in a sorted block is a single operation **Join** Finding a value within a block can be done via binary search.

Figure 3: Optimizations on Compressed Data

In summary, by using *compressed blocks* as an intermediate representation of data, operators can operate directly on compressed data whenever possible, and can degenerate to a lazy decompression scheme when this is impossible (by iterating through block values). Further, by abstracting general properties about compression techniques and having operators check these properties rather than hard coding knowledge of a specific compression algorithm, operators are shielded from needing knowledge about the way data is encoded.

实验结果

- 测试环境
- 测试方法
- 术语

测试环境

- 3.0 GHz Pentium IV
- RedHat Linux
- 2 Gbytes of memory
- 1MB L2 cache
- 750 Gbytes of disk ， 50-60MB/sec
- synthetically generated
- TPC-H data, lineitem 10X,60,000,000

测试方法

	Synthetically Generated data	TPC-H Data	说明
Eager Decompression	是指从磁盘上读上来的压缩页直接进行解压后进行访问操作		观察到影响性能的两个主要因素： 1. the number of distinct values; 2. sorted run lengths
Operating Directly on Compressed Data	是指从磁盘上读上来的压缩页，直接在压缩态数据上进行操作访问，或者 lazy decompress		
Higher column cardinalities	Queries ran with competition for CPU cycles 在有 CPU 竞争的情况下进行查询测试		

简单查询语句为： SELECT SUM(C) FROM TABLE GROUP BY C;

术语

- *Cardinality:* mean the number of distinct values
- Sorted runs of size: 主要是指排序后相同值可能重复的次数
- dictionary single-value: 从整数中抽取符号；对符合分组统计；解压后相乘得到总和；
- dictionary multi-value: 统计相同整数的个数；不解压相乘得到总和；

Eager Decompression

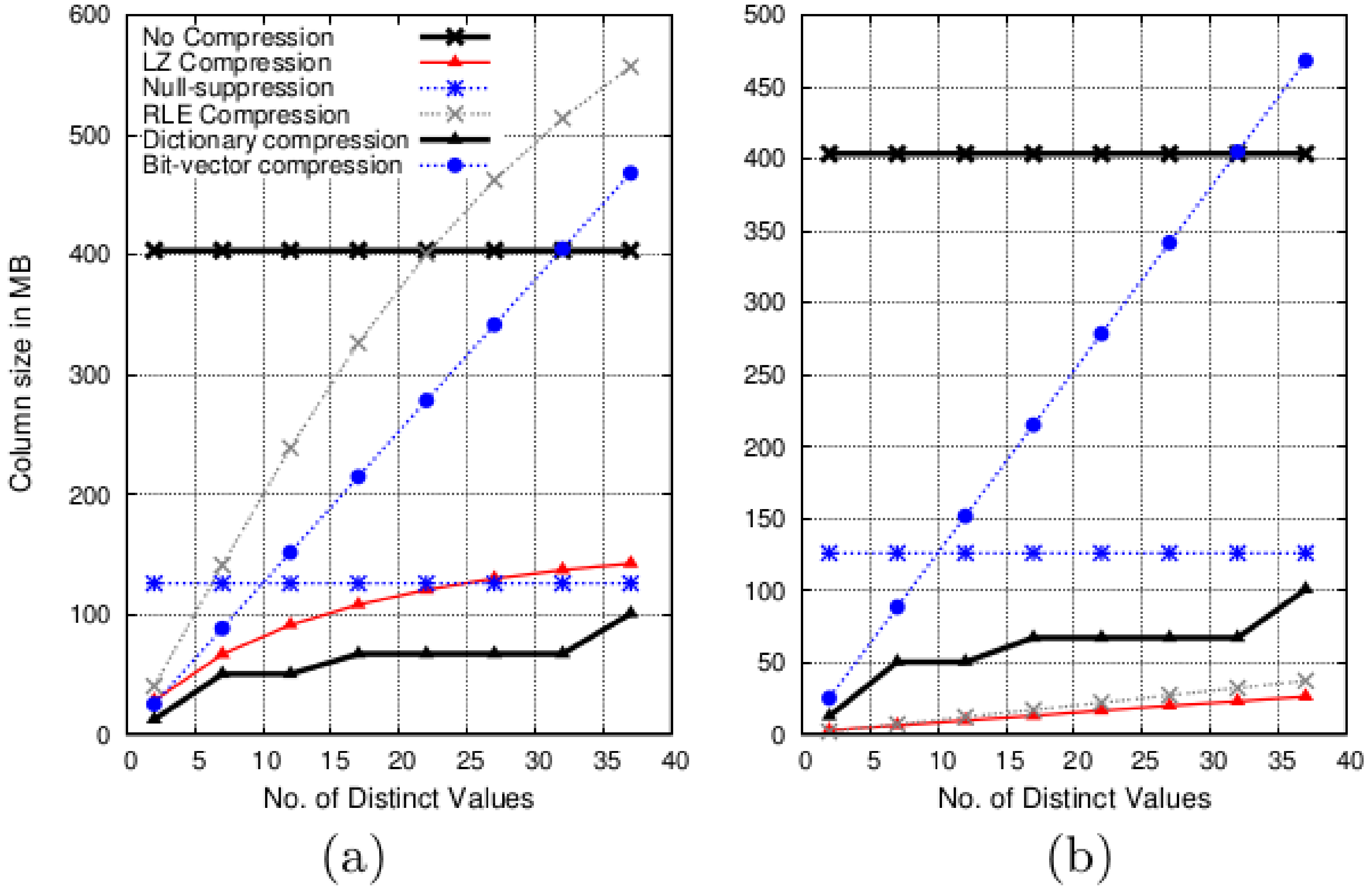


Figure 4: Compressed column sizes for varied compression schemes on column with sorted runs of size 50 (a) and 1000 (b)

1 ）字典压缩和 LZ 压缩效果最好；
2 ）在相同值重复度大的情况下，RLE 压缩有很好的效果（见图右）
3 ）bit vector 无进一步压缩，且唯一值个数不变，效果相同

Eager Decompression

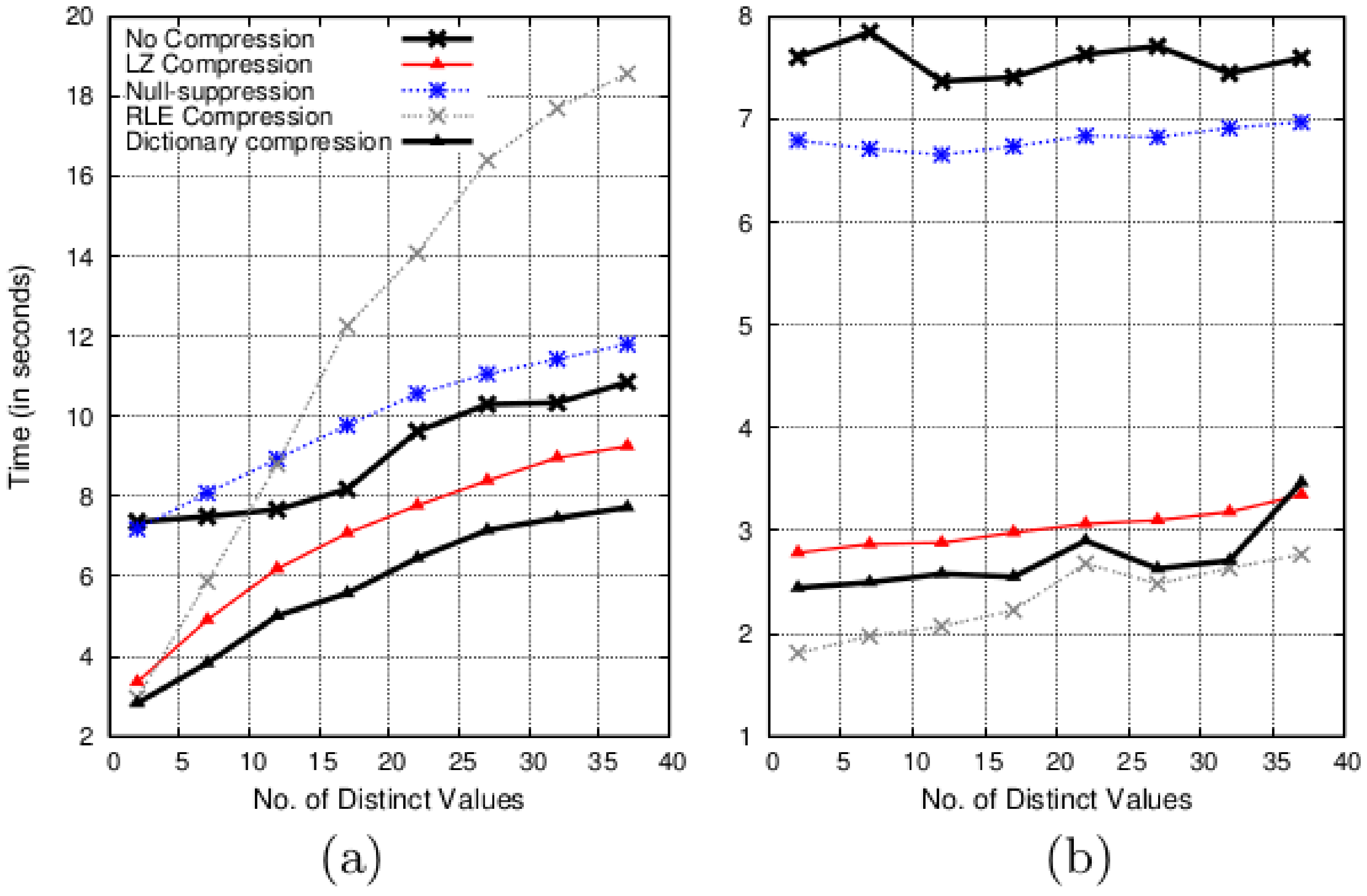


Figure 5: Query Performance With Eager Decompression on column with sorted runs of size 50 (a) and 1000 (b)

1）磁盘上的大小不是查询性能的主要因素，为 CPU bound ；
2）bit vector 方法性能太劣，没有显示在图中；
3）字典和 LZ 解压性能优于 RLE 和 NS ；
4）RLE 在 run length 提升的情况下解压性能也提升；
5）RLE/NS 解压过程中 if/else 影响性能；

Operating Directly on Compressed Data

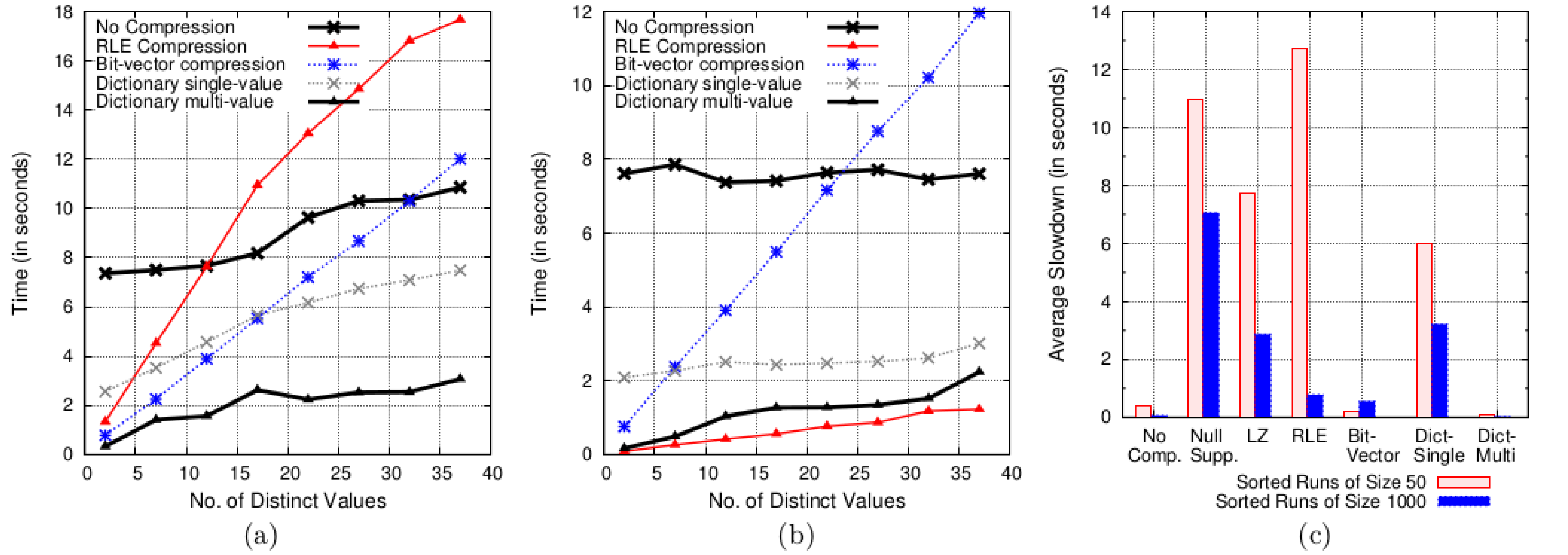


Figure 6: Query performance with direct operation on compressed data on column with sorted runs of size 50 (a) and 1000 (b). Figure (c) shows the average increase in query time relative to the query times in (a) and (b) when contention for CPU cycles is introduced.

1 ）LZ 和 NS 这两种方法是无法进行 operate on compressed data 的 ；
2 ） CPU 方面的提升是查询性能提升的最主要原因，而非 cache line 方面；
3 ）operating directly on compressed data reduces both I/O and CPU cycles

Higher column cardinalities

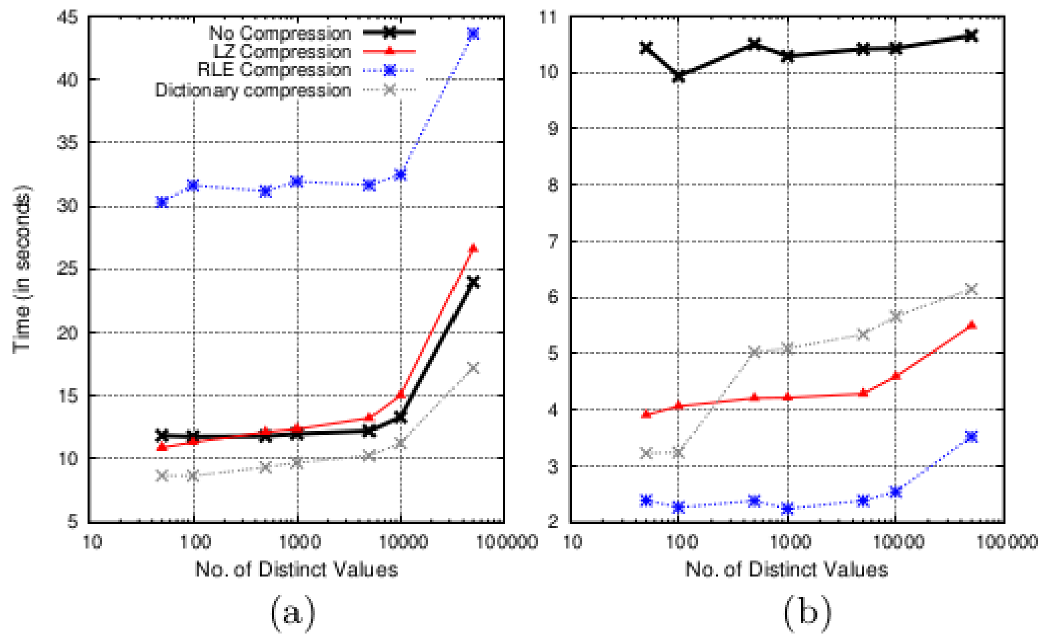
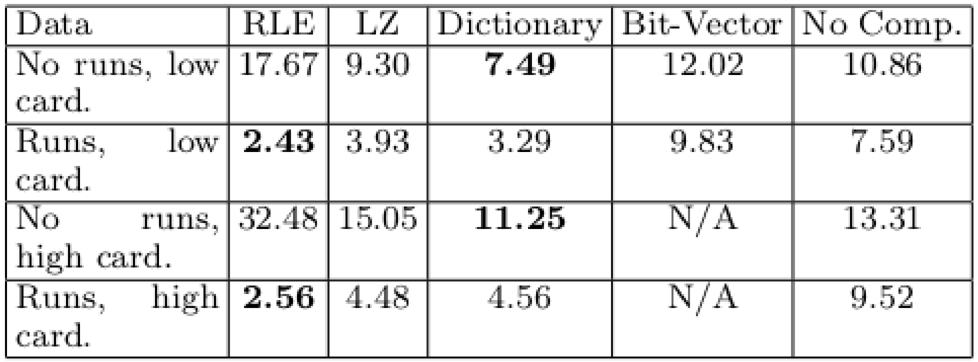


Figure 7: Aggregation Query on High Cardinality Data with Avg. Run Lengths of 1 (a) and 14 (b)



Data	RLE	LZ	Dictionary	Bit-Vector	No Comp.
No runs, low card.	17.67	9.30	**7.49**	12.02	10.86
Runs, low card.	**2.43**	3.93	3.29	9.83	7.59
No runs, high card.	32.48	15.05	**11.25**	N/A	13.31
Runs, high card.	**2.56**	4.48	4.56	N/A	9.52

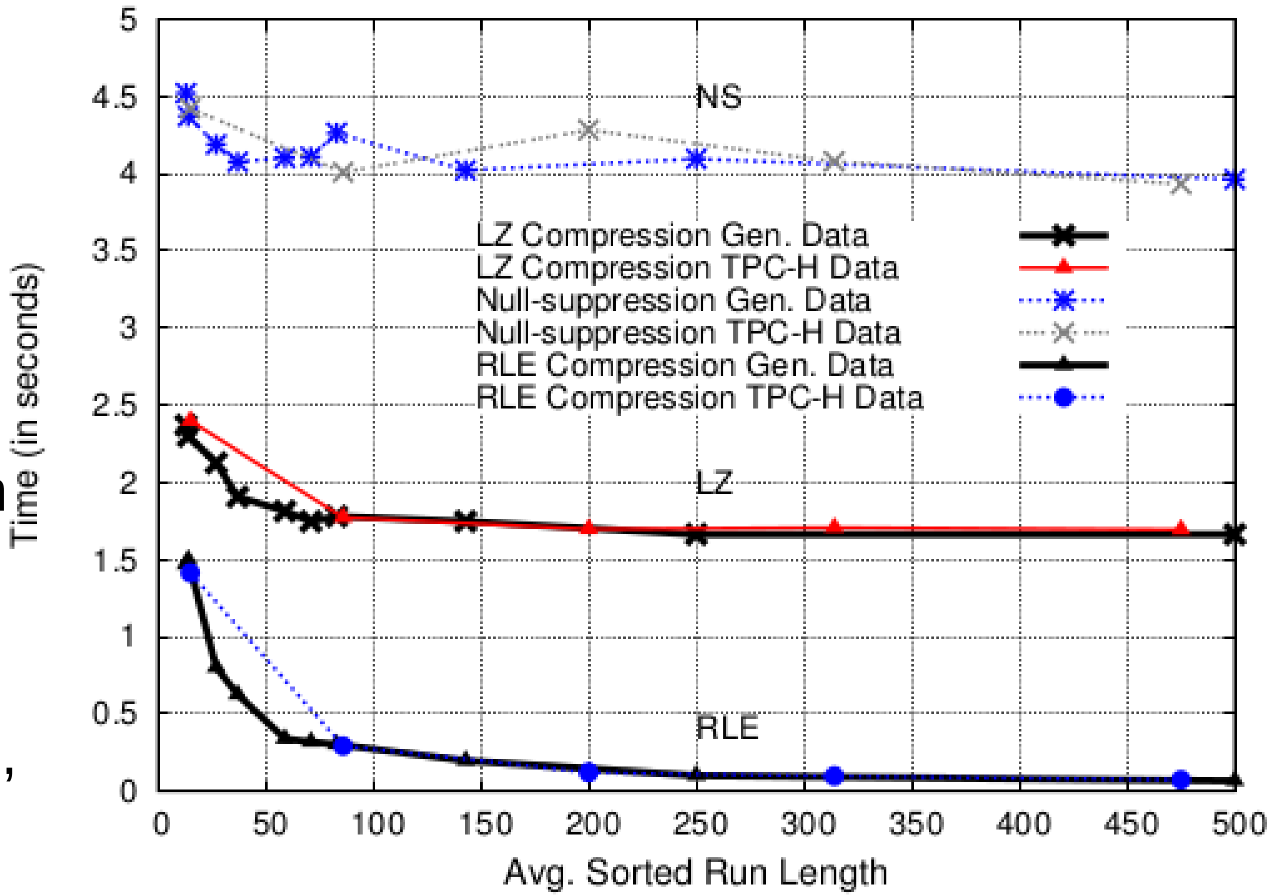
1）使用的是均匀分布数据，即数据随机分布性大；
2）字典压缩对于随机数据分布的查询性能相对表现较好；
3）RLE/LZ 的 data location 好，但是不适用于随机数据分布；在 run length 提升的情况下，是可以有性能提升的；

右边的表时间为秒
1）summarize how data characteristics affect aggregate query performance on various compression types;
2）对于 RLE 和 LZ 而言，run length 是比 cardinality 更重要的影响性能的因素；

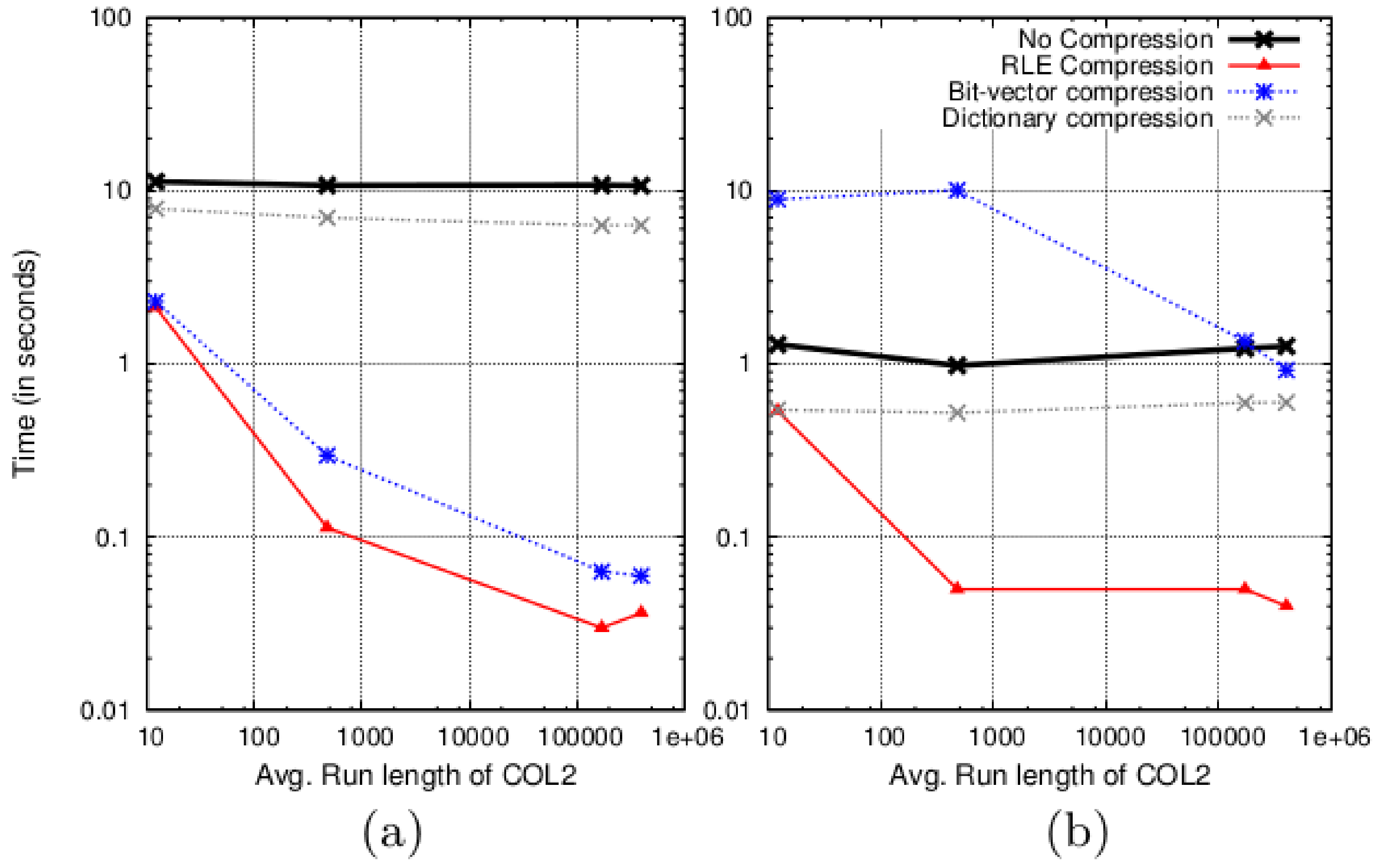
Generated vs. TPC-H Data

(shipdate, retflag, quantity) [314]
(price, retflag) [15]
(suppkey, linenumber) [86]
(suppkey, retflag) [200]
(shipdate, quantity) [475]

1 ）上面表的所有 projection 都进行排序；
2 ）run-length is a good predictor of query performance for the RLE, LZ, and null-suppression compression schemes except bit-vector and dictionary



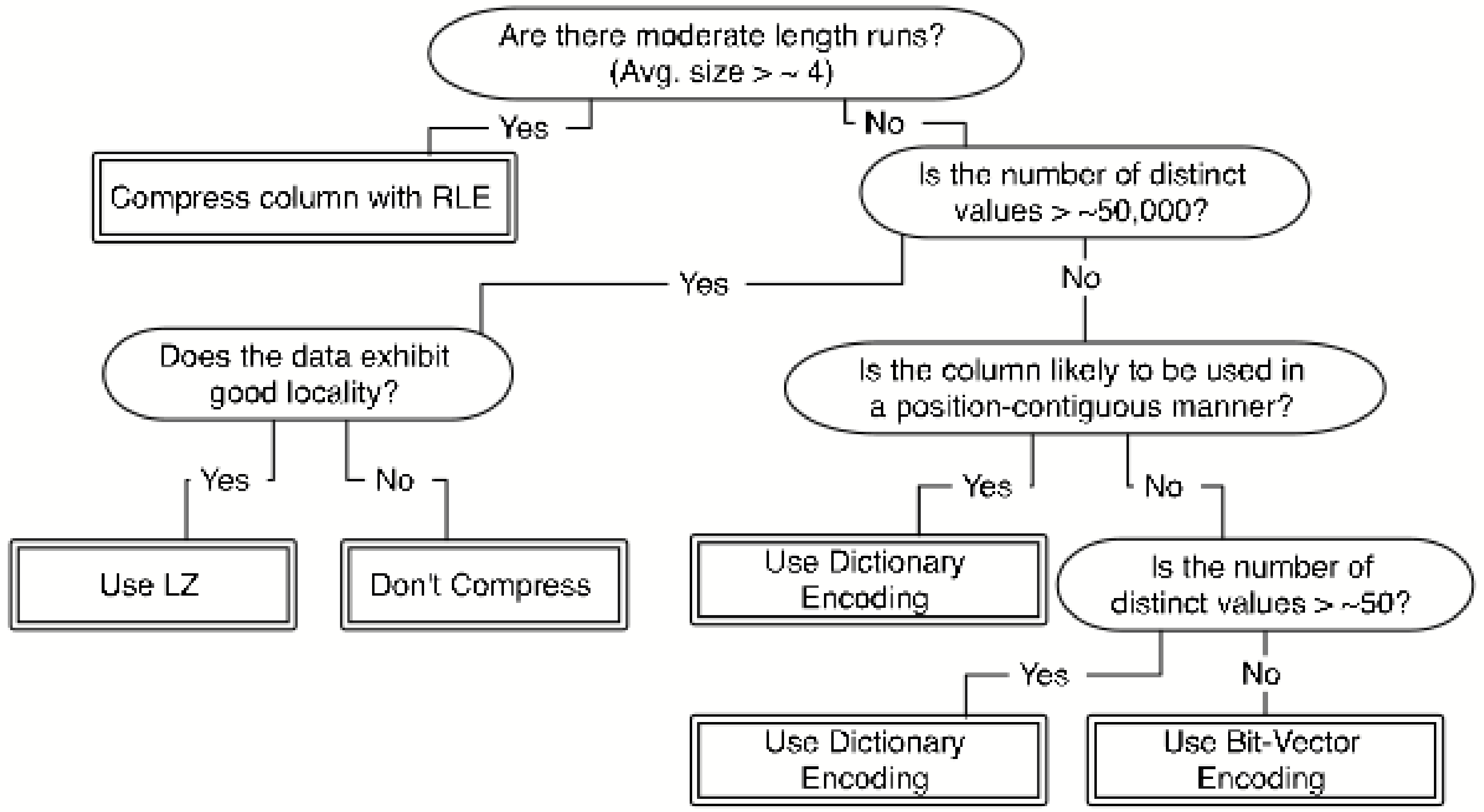
Other Query Types



SELECT COL1,
COUNT(*)
FROM
CSTORE_PROJ1
WHERE
PREDICATE(COL2)
GROUP BY COL1;

SELECT S.COL3,
COUNT(*)
FROM CSTORE_P1
AS L, CSTORE_P2
AS S
WHERE
PREDICATE(S.COL2
) AND
PREDICATE(L.COL1
)
AND
L.COL2=S.COL1
GROUP BY S.COL3;

结论



Are there moderate length runs? (Avg. size > ~ 4)
Yes
No
Compress column with RLE
Is the number of distinct values > ~50,000?
Yes
No
Does the data exhibit good locality?
Is the column likely to be used in a position-contiguous manner?
Yes
No
Yes
No
Use LZ
Don't Compress
Use Dictionary Encoding
Is the number of distinct values > ~50?
Yes
No
Use Dictionary Encoding
Use Bit-Vector Encoding

Figure 10: Decision tree summarizing our results regarding the proper selection of compression scheme.

结论

- Physical database design should be aware of the compression subsystem 压缩系统需要作为物理数据库设计的一部分去考虑
- Sacrificing the compression ratio of heavy-weight schemes for the efficiency light-weight schemes in operating on compressed data is a good trade-off to make 牺牲一定的压缩比来换取好的查询性能
- cost models that only take into account I/O costs will likely perform poorly in the context of column-oriented systems since CPU cost is often the dominant factor 代价模型在考虑IO 代价的同时也需要考虑 CPU 代价