Summer 2019

Efficient Query Execution over Large Databases through Semantic Caching of Bitmap Indices

Sarah McClain
University of Puget Sound

Efficient Query Execution over Large Databases through Semantic Caching of Bitmap Indices

Sarah McClain and David Chiu (advisor)
Department of Mathematics and Computer Science, University of Puget Sound

Background

As popular applications become increasingly data-intensive the need for novel techniques to query large-scale data stores becomes more prevalent. Because computers' hard drives are slow, the more data is stored, the longer it takes to access useful information. For this reason, it is imperative to use efficient data structures such as bitmap indices and caches to provide fast data access.

What is a bitmap index?

A bitmap index is a way of consolidating a large data set by representing certain columns using a single bit (0=false, or 1=true) for each row. The data is then represented by expanding the columns, or *attributes*, into what are called *bins* and represent the value of the row, or *tuple*, by a 0 or 1 depending on whether it falls in a bin.

Tuple	Name	Sex	Age
t1	Kelsie	F	9
t2	Megan	F	26
t3	Nalin	M	21
t4	Zac	M	80
...	...	...	...

Sample Database

Tuple	Name				Sex		Age		
	Kelsie	Megan	Nalin	Zac	M	F	0-18	19-50	51-99
t1	1	0	0	0	0	1	1	0	0
t2	0	1	0	0	0	1	0	1	0
t3	0	0	1	0	1	0	0	1	0
t4	0	0	0	1	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...

Corresponding Bitmap Index

What is a semantic cache?

Due to limited space on the (CPU), most data is stored in secondary storage, such as the disk, which is very slow. A semantic cache is a form of architecture that caches the result of selection queries after being fetched from the disk. Semantic caching involves splitting a query into two pieces, a *probe query* and a *remainder query*. The probe query addresses cached regions that contribute to the new query. The remainder query refers to data not contained in the cache (but relevant to the query) that much be fetched from secondary memory. Semantic caching allows the computer to reduce access to the disk by reusing partial matches in the cache.

Mechanism

Select * from users where age >= 15 AND age < 75

The bins for age correspond to columns 20 through 26 for this bitmap. The SQL query is looking to retrieve columns 21 through 25 of the bitmap and execute an "OR" operation to get all the users between ages 15 and 75. The cache contains already columns [21,23].

Column #	20	21	22	23	24	25	26	...
...	Age (years)							...
...	<15	15-29	30-44	45-59	60-74	75-99	>99	...
...	1	1	0	1	1	1	1	...
...	0	1	0	1	0	1	0	...
...	0	0	1	0	0	0	0	...
...	1	1	1	1	0	1	1	...

1 Probe Query

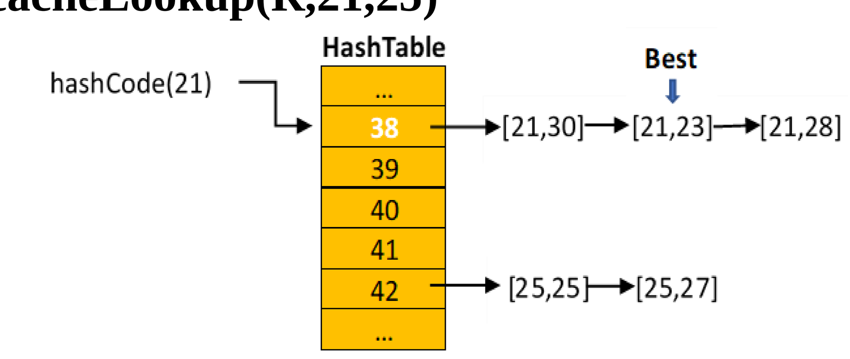
Select * from users age >= 15 AND age < 60

Remainder Query 3

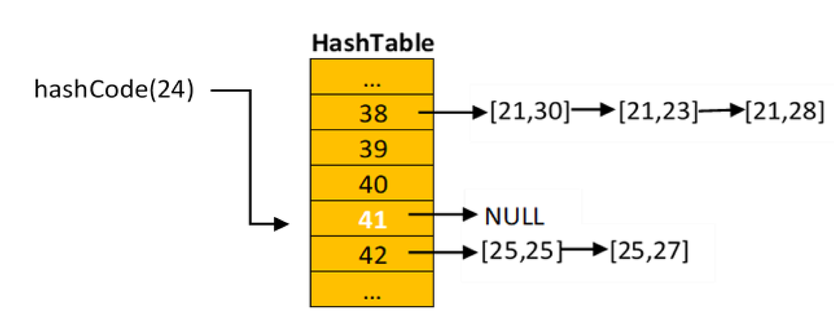
Select * from users where age >= 60 AND age < 75

Cache

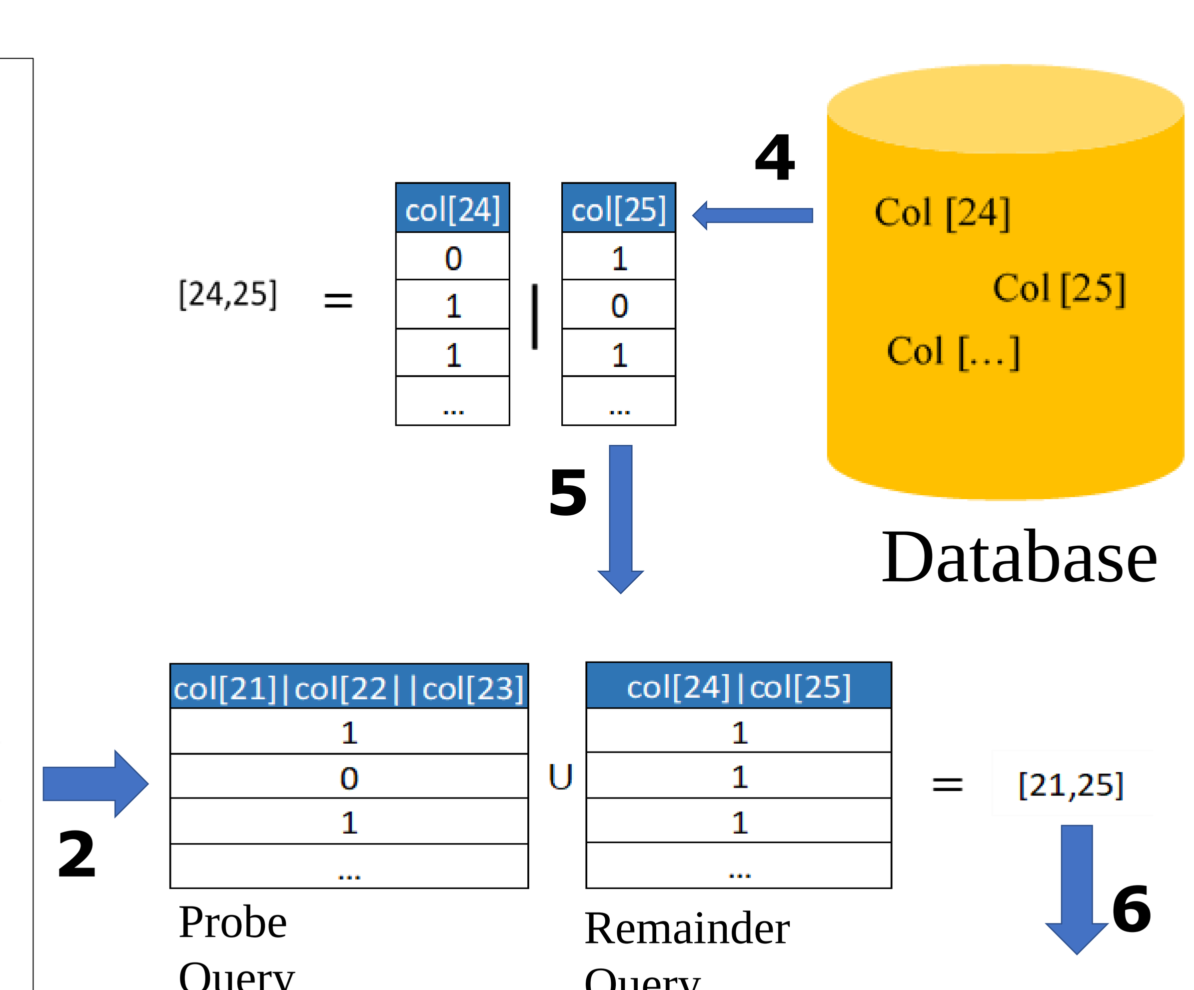
cacheLookup(R,21,25)



R = getData(21,23)
cacheLookup(R,24,25)



Return end_Q = 23



Return [21,25]

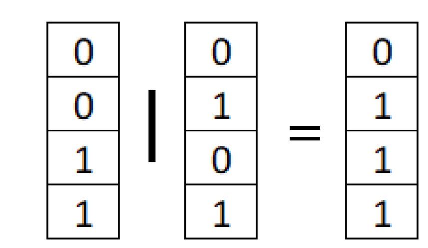
Query File

What is an "OR" operator?

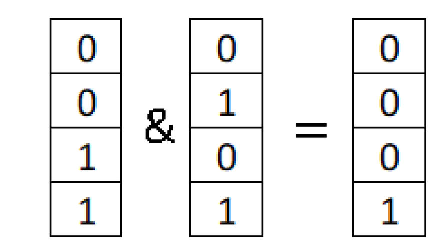
An "OR" operator (denoted by the "|" symbol) is a Boolean operator that returns true (1=true, 0=false) if either (or both) operands is true

What is an "AND" operator?

An "AND" operator (denoted by the "&" symbol) is a Boolean operator that only returns true if both operands are true.



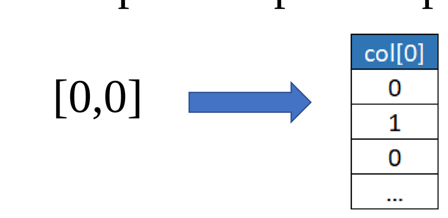
OR operation



AND operation

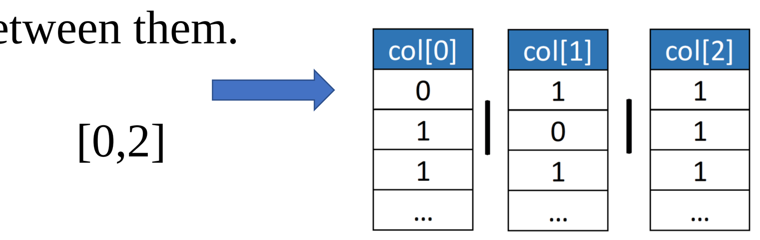
What is a point query?

A point query is searching for a single column in the bitmap. We represent point queries as such:



What is a range query?

A range query is retrieving data for multiple columns and computing an "or" operation between them.



Algorithm

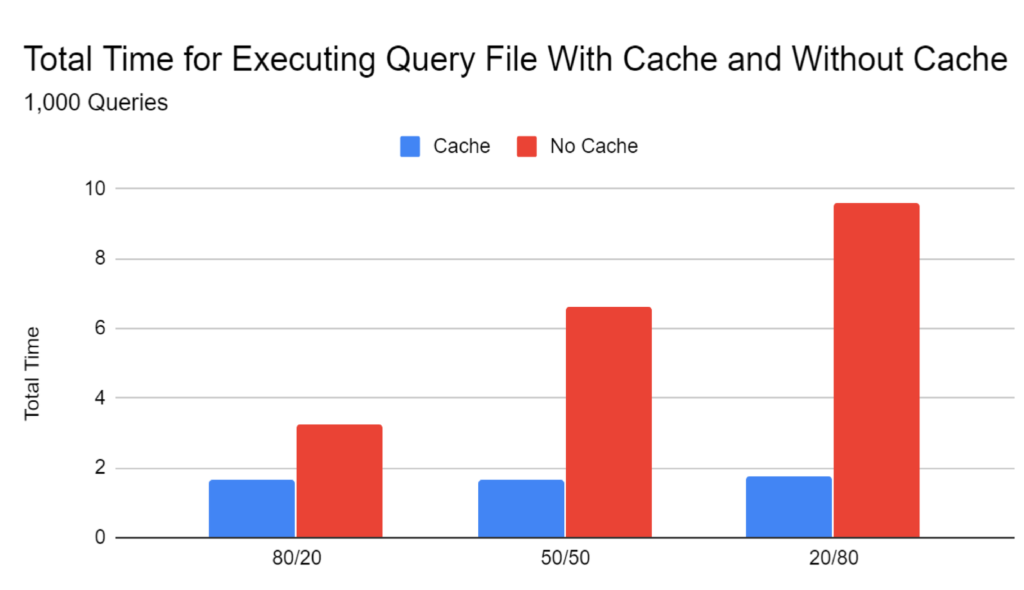
Algorithm 1 $cacheLookup(R, start_Q, end_Q)$

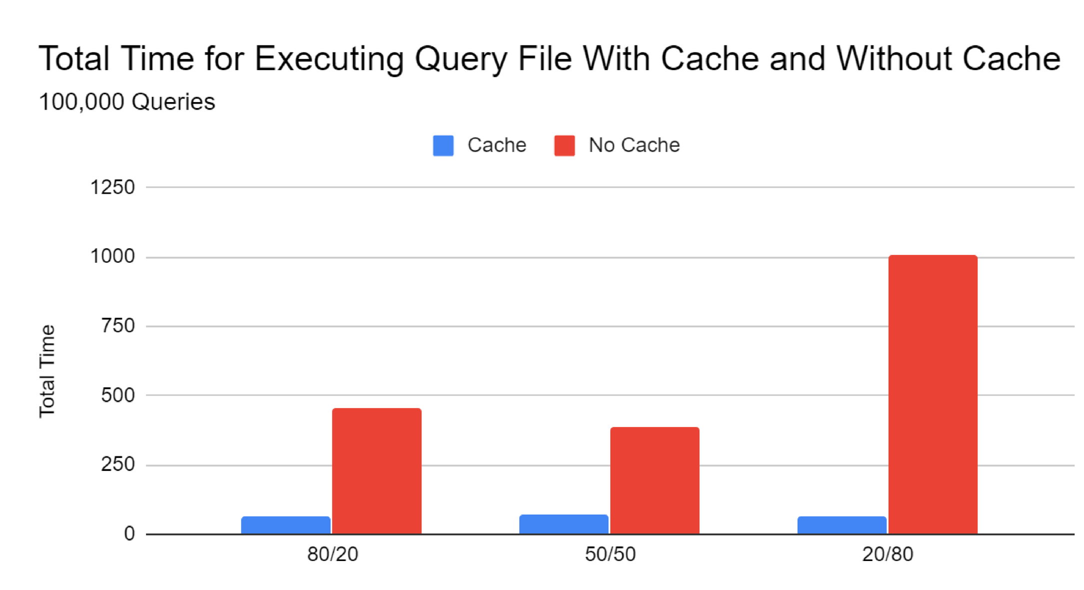
Input
- R — Set of result vectors
- $start_Q$ — the starting column of full query
- end_Q — the ending column of full query

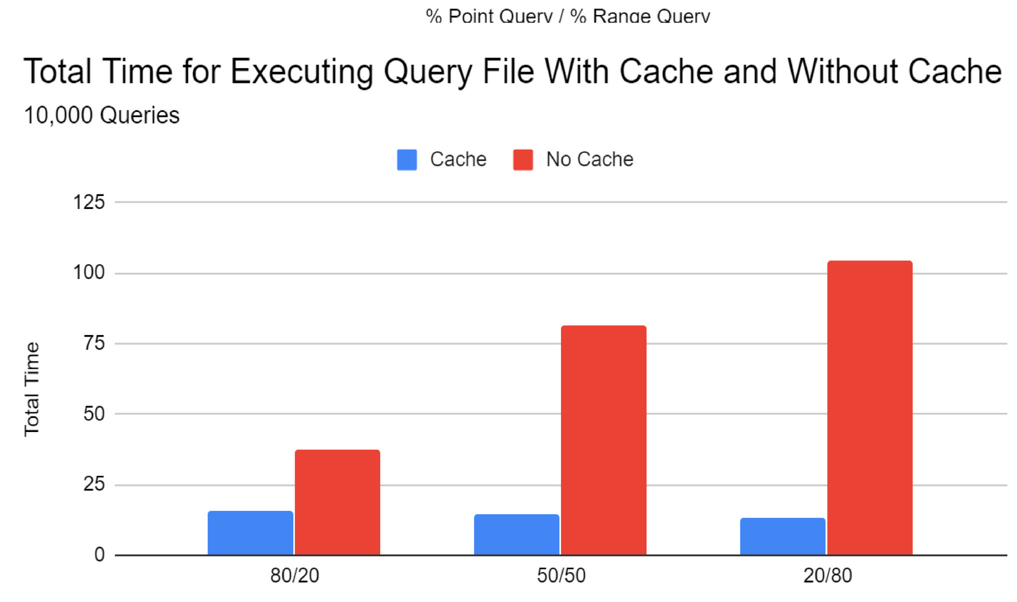
Output
- R^* — Result vectors for probe query $[start_Q, end'_Q]$
- end_Q — ending column of probe query

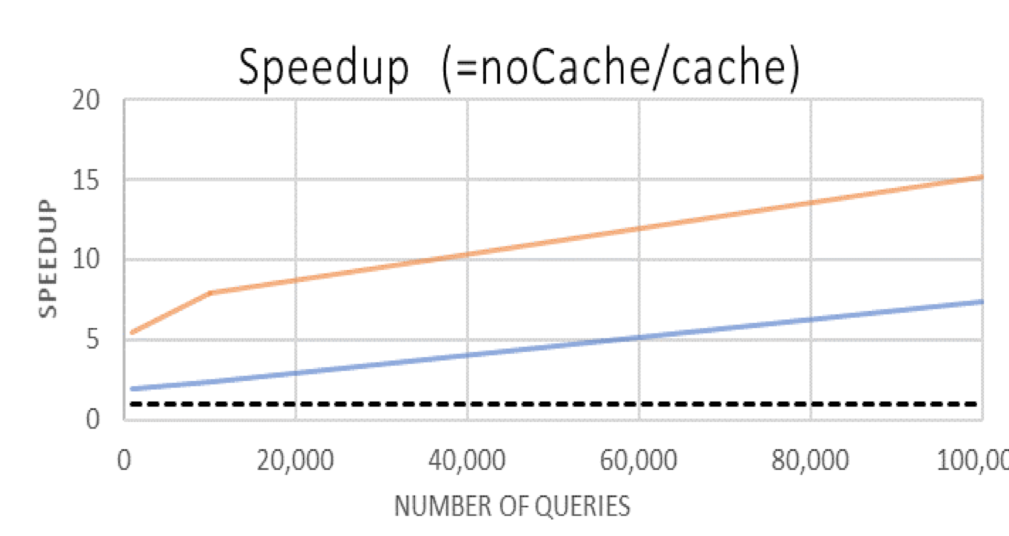
i $\leftarrow hashCode(start_Q)$
best $\leftarrow$ list[i].head
for all $[start_Q, end'_Q]$ in $list[i]$ **do**
  if $end_Q = end'_Q$ **then**
    $R \leftarrow getData([start_Q, end_Q])$
    $R^* \leftarrow R \cup R^*$
    return end_Q
  if best.difference $> |calculateDifference(end'_Q, end_Q)|$ **then**
    best $\leftarrow [start_Q, end'_Q]$
    best.difference $= |calculateDifference(end'_Q, end_Q)|$
if $end'_Q \geq end_Q$ **then**
  $R \leftarrow getData([start_Q, end'_Q])$
  $R^* \leftarrow R \cup R^*$
  return end'_Q
return $cacheLookup(R, end'_Q + 1, endQ)$

Plots









Results

The semantic cache tremendously decreased the amount of time needed to query the database. As shown in the figures, as the number of queries grows, it becomes more efficient to leverage this caching architecture. The semantic cache also has a greater speedup as the number of range queries increases. There is still more to be studied about semantic caching. More work on the effect of different lookup algorithms and caching compound ranges could increase efficiency even more.

Acknowledgements

I would like to thank the University of Puget Sound for providing me the opportunity to complete this project as well as the resources to print and present my research. I would also like to give a special thanks to my advisor, David Chiu, for his time and support as he guided me throughout the course of this project.