Chapter 14
Scan Performance

14.1 Introduction

In this chapter, we discuss the performance of the scan operation. Scan operations require the values of a single attribute or a small set of attributes but go through the whole dataset. Scan operations search one or more attributes for a certain value. Unless the table is sorted by the attribute to scan, a scan has to iterate over all lines and returns those lines, which fulfill the search predicate (e.g. "SELECT * FROM world_population WHERE lastname = 'Smith'"). As in Chap. 13, we will discuss the influence of different layouts (row and column) and different approaches on the performance of the scan operation. We compare the following three approaches:

- full table scan in row layout
- stride access for the selected attributes in row layout
- full column scan in columnar layout

In the following examples, the world population table already known from previous chapters is scanned. To recap, the table has the following properties:

- 8 billion tuples
- tuple size of 200 byte
- table size of 8 billions · 200 byte = 1.6 TB
- attributes: first name, last name, gender, country, city, birthday
- all attributes have a fixed length

In addition, the previous assumptions for the response time calculations are used:

- bandwidth of read operations from main memory: 2 MB/ms/core
- cache line size of 64 byte

In the first example, the scan operation will help to answer the question: "How many women are in the world?".

The target column that has to be scanned to answer this question is "Gender", which has two possible distinct values. For simplicity, the calculations of the scan performance are done using a single core. When performing the scan operation,

H. Plattner, *A Course in In-Memory Data Management*,
DOI: 10.1007/978-3-642-36524-9_14, © Springer-Verlag Berlin Heidelberg 2013

each row of the table is independent from all other rows. Consequently the scan operation can be efficiently parallelized and scales nearly linearly.

14.2 Row Layout: Full Table Scan

Having the data organized in a row layout, the first and most obvious approach to find the exact number of women in the world is to scan sequentially through all rows to read the gender attribute. We have seen this behavior for software that uses Object-Relational Mapping (ORM) and does calculations on the application side. Having to retrieve whole data sets in order to create the needed objects to interact with, results in a full table scan. During this operation, the CPU will read 1.6 TB from main memory. Taking into account the scan speed of 2 MB/ms per core, we can calculate the runtime on one core as follows:

$$\textit{Full table scan response time on } 1 \textit{ core} = \frac{1.6 \text{ TB}}{2 \text{ MB/ms}} = 800 \text{ s}$$

We would have to wait for more then 10 min to get the answer to our question. In order to achieve a better performance we have to look for optimizations. An obvious and simple solution is to compute the question in parallel on multiple cores and CPUs. We could do a vertical partitioning of the table and let the processing units execute the scan operation on the table parts in parallel.

Let us have a quick look on an example for a quad core CPU for that. The scan speed for a quad core CPU can be calculated as follows: 4 cores · 2 MB/ms/core = 8 MB/ms. The full table scan response time is 1.6 TB/8 MB/ms = 200 s. Even with four cores the query execution takes several minutes.

Another approach that could help to increase the performance of the scan operation is to take advantage of the in-memory database and read the gender fields with direct access. On disk based databases, only pages instead of single attributes are usually directly accessible (see Sect. 4.4). The results for this approach are calculated and discussed in the next Sect. 14.3.

14.3 Row Layout: Stride Access

Assume we still use row layout to store the data in memory. But now, instead of scanning the whole table and reading all table fields from main memory, we read the target field via direct access. To scan all gender fields the CPU does 8 billions cache accesses, one access for each tuple. Assuming the cache line size is 64 byte, and considering the fact that a CPU will read exactly 64 byte during each cache access independently of the gender field size, we can calculate the data size that is read from main memory during the whole scan operation:

$$data\ volume\ =\ 8\ billion \cdot 64\ byte \approx 512\,\text{GB}$$

Taking into account the scan speed, we get the following the response time for one core:

$$Stride\ access\ response\ time\ on\ 1\ core = 512\ \text{GB}\ /\ 2\ \text{MB/ms} = 256\,\text{s}$$

The result is better than that for the full table scan, but answering the question still takes several minutes. However, there are further opportunities to optimize the scan speed for our initial question.

The next Sect. 14.4 will discuss the effects of using a columnar data layout.

14.4 Columnar Layout: Full Column Scan

When using a columnar layout, the data is stored attribute-wise in main memory. This fact leads us to the following conclusions:

- As attributes of the same types are stored together, effective compression algorithms can be used to reduce the data volume that is stored in memory and that has to be transferred between main memory and CPU.
- As values of the same attribute are stored consecutively, the probability that the next accessed item has already been loaded as part of the same cache line, is relatively high depending on the length of the compressed values. The shorter the values, the higher the probability.

Consequently, two aspects of columnar layouts can be leveraged: scanning only target fields and reading compressed values. Both aspects reduce the data volume transferred between main memory and CPU and consequently reduce response times. In our example, the CPU will scan through the "Gender" field.

Considering this column is dictionary-encoded as described in Chap. 6, only one bit is necessary to encode the two possible values 'm' and 'f'. As before, we can calculate the data volume to be read from main memory using the size of the attribute and the number of tuples: 8 billion · 1 bit ≈ 1 GB, which leads to a full column scan response time of 1 GB/2 MB/ms/core = 0.5 s on one core.

The result shows a significant difference in the performance in comparison with both presented approaches for the row layout. Further taking into account the opportunity to use several cores and to execute the scan operation in parallel, using the column layout we can speed up the answer to our example-question even more. While our example query, which is using only one attribute and is posed against a vast number of tuples might not be the common use case, it can be stated that in analytical workloads the general circumstances are favorable for this approach, as we have already seen in Chap. 3. Queries against huge data volumes that operate on a small number of columns are characteristic for analytical and transactional enterprise applications.

14.5 Additional Examples and Discussion

In our previous example, we considered an almost "perfect" case. With 1-bit length, the gender attribute was compressed to the minimum for dictionary-encoded values. This fact decreased the data volume to be transferred between CPU and main memory. Of course, the outcome of the performance calculation depends on the size of the scanned fields. For larger fields, the CPU will need to scan through a higher data volume and less values will fit into one cache line.

To compare the results with another attribute, let us take the same table from the first example and calculate the response time for the full column scan operation on the column "Birthday". This column has more distinct values than the "Gender" field.

Considering that every value (i.e. valueID of a compressed value in the "Birthday" column) has a size of 2 byte, we can calculate the transferred data volume and appropriate response time as follows:

- data volume to be read from main memory = 8 billion · 2 byte ≈ 16 GB
- full column scan response time = 16 GB/ 2 MB/ms/core = 8 s (with one core)

To summarize the calculations performed above, we can conclude, that the following parameters of a CPU and a scanned table influence scan performance:

- cache utilization
- memory bandwidth
- number of processing units
- number of tuples in the table (table cardinality)
- used compression
- used layout: column or row layout

The example calculations in this chapter show a significant speed up of the scan performance when switching from row to dictionary-encoded column layout. While the columnar layout with its higher data density better utilizes the CPU caches, we would also like to note that it enables further optimizations, e.g. the usage of SIMD/SSE operations (see Sect. 17.1.2).

14.6 Self Test Questions

1. **Loading Dictionary-Encoded Row-Oriented Tuples**
   Consider the example in Sect. 14.2 with dictionary-encoded tuples. In this example, each tuple has a size of 32 byte. What is the time that a single core processor needs to scan the whole world_population table if all data is stored in a dictionary-encoded row layout?

   (a) 128 s
   (b) 256 s
   (c) 64 s
   (d) 96 s