Bloom filters

Toon Koppelaars
Real-World Performance, Oracle Server Technologies
Netherlands

Keywords:

Bloom filter, parallel query, hash join, Bloom pruning

Introduction

"A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set."
From: https://en.wikipedia.org/wiki/Bloom_filter

The concept of a Bloom filter is not new, nor is its use by the cost-based optimizer (CBO) as the CBO can make use of them ever since Oracle10G release 2. However still today, many database engineers are not familiar with Bloom filters, in the sense of what they intend to achieve and how they go about doing that. In this presentation we will give a detailed report of exactly these two topics.

We will make you familiar with the concept of a Bloom filter and show you various usages of them by the CBO through discussion of example queries and their execution plans which use Bloom filters. Since Bloom filters will only ever show up in hash-joins, we will start with a discussion of this join type, and more specifically make you familiar with reading a parallel execution plan of a hash-join. Since both hash-joins and Bloom filters depend on the concept of a "hash-function", we'll start the presentation by explaining what a hash-function is.

Hash functions and hash joins

Hash functions transform byte-strings of arbitrary length into a numeric value (the "hash value") that represents an offset typically in the range of integers starting at zero up to some power of two. Applying a hash function is cheap (by design), and they are supposed to work such that the hash-values themselves are evenly distributed in the range, irrespective of the distribution of the input byte-strings.

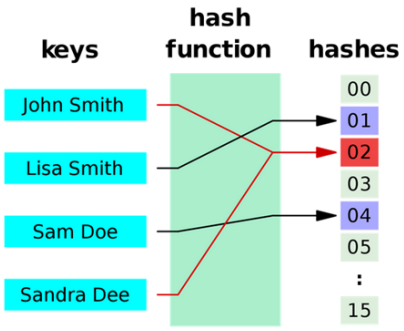

Illustration 1: Hash Functions

As shown in illustration 1, collisions can occur: that is, two different input values might result in the same hash-value. For joining many rows the CBO would typically employ the hash-join method. In this join method the smaller table will be read and "hashed" into a hash-table by applying a hash-function on the join column(s). This is done during the "build-phase" of a hash-join. Then during the "probe-phase" of a hash-join the other table will be scanned and for each row the same hash-function will be applied to the join column(s). Using the hash-value the hash-table can be probed to find rows of the smaller table that join to the current row of the 2nd table (see illustration 2).

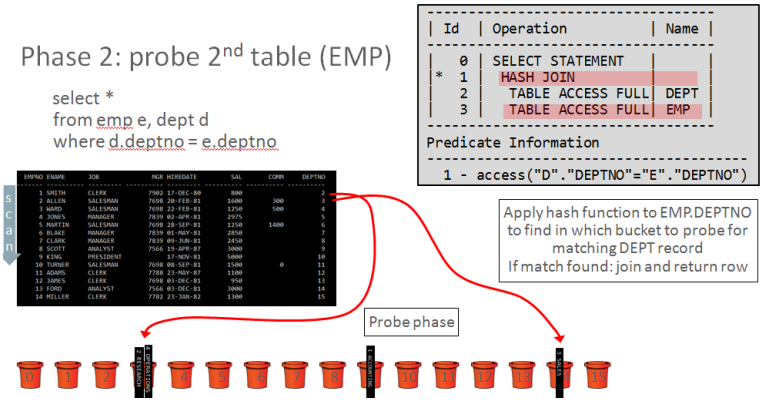

Illustration 2: Serial Execution of a Hash Join

Parallel execution of a hash join

Since many rows are typically joined when performing a hash join, parallelization of this join type is a good means to speed-up the execution. The parallel execution of a hash join employs two sets of parallel query (PQ) slave sets, which in turn work on the build and probe phases of a hash-join. At the same time for each phase this work is performed in parallel, by distributing the rows of both tables in a certain way over the PQ processes. In the presentation we'll discuss the hash-distribution, which also employs a hash-function.

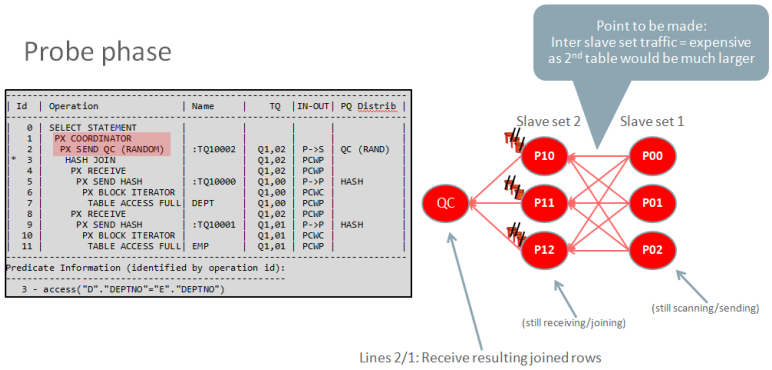

Illustration 3: Parallel Execution of a Hash Join

Allthough the work is performed in parallel, still one of the main bottlenecks of this parallel execution of a hash join, is what's called the "inter-slave traffic". All rows need to be communicated from one PQ slave set to the other PQ slave set during the probe phase. And this is where Bloom filters enter the stage.

Early filtering through Bloom filters

As we'll explain during the presentation, a Bloom filter can be used to represent, in a mathematically smart way, the set of join-values. During the build phase of a hash-join all join column values from the 1st row source will be "stored" in a Bloom filter. Then during the subsequent probe phase of the hash join, this Bloom filter will be used to prevent the sending of rows from the 2nd rows source to the other slave set that would definitely not find a join partner in the hash table of the 1st row source.

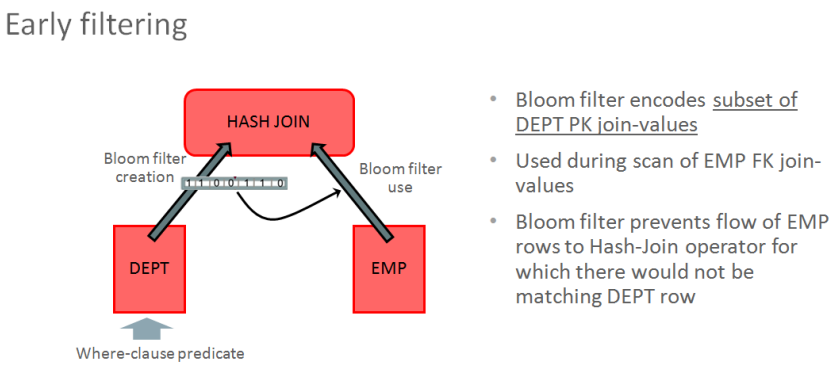

Illustration 4: Bloom filter creation and use during a hash-join

And by doing so, a Bloom filter, whose initial creation and subsequence application is fairly cheap, can result in a dramatic decrease of inter slave-set communication. We'll demonstrate this by discussing a SQL Monitor report of a parallel hash-join with and without application of a Bloom filter.

Contact:

Toon Koppelaars
Consulting Member of Technical Staff
Real-World Performance, Oracle Server Technologies

Email toon.koppelaars@oracle.com
Internet: Twitter @ToonKoppelaars