Ferhat Elmas 214805

Advanced Databases Hw 5

1- Bloom Join:

Explanation:

Main join site:

- Compute the hash values of the join column in a bit vector k.
    - Map values in join column between 0 to k-1.
- Send this bit vector to the other site.
- Take the other relation from other site.
- Apply one of the standard join algorithms such as merge-join or hash-join.

Other site:

- Take the bit vector from main site.
- Compute the hash values of the join column in a bit vector of same sized.
- Select tuples according to these hashes and taken bit vector.
- For example, values {A, B, C, D, E, F} and bit vector size of 3:

○ Main site:		Other site:
○ A → 0		C → 1
○ B → 0	→ 110 →	E → 2
○ C → 1	Bit vector	
○ D → 1		

- Here, E is mapped to 2 but 2 doesn't exist in the first relation since in the bit vector, 110, third bit is zero. Therefore, even if we send the tuple that has E in join column, main site would eliminate this tuple and it wouldn't be in the result set so this site doesn't send this tuple to decrease network communication. In above example, only first tuple, C, is sent to main site.

Pseudo Code:

Main join site:

```
Input: k bit vector size
Output: join result
Processing:
        bit vector ← create a zero bit vector of length k
        for all tuples in the relation:
                hash ← hash the value in join column into [0, k-1]
                bit vector[hash] ← 1
        end
        send bit vector to the other site
        reduced tuples of other relation ← take tuples of the other relation
        result ← make join with local relation and reduced tuples of other relation
        return result
end
```

Other join site:

```
Input: k bit vector size
Output: nothing
Processing:
    bit vector ← take bit vector from main site
    reduced ← [ ]
    for all tuples in the local relation:
        hash ← hash the value in join column into [0, k-1]
        if bit vector[hash] = 0
            skip this tuple
        else
            reduced ← reduced + this tuple
        end
    end
    send reduced to main site
end
```

2- Data sets get bigger and efficient analysis of these data sets is required. Map-reduce framework makes this possible on commodity hardware. However, development directly for this framework is low-level and procedural so solutions are custom to the problems that makes the reuse problematic. Therefore, we need an abstraction and we should be able to submit jobs in a declarative paradigm. However, most programmers come from procedural background and declarative style can be restrictive. Pig makes the balance and services these needs by improving development and execution speed.

3- Pig puts filter and foreach commands until first (co)group command into mapper. Then, a job is created for each (co)group. Filter and foreach commands in between are put into reducer of (co)group commands. Therefore, for D, a job is created and A, B and C are put into map primitive of this job. D is basically shuffling. E goes into reducer of the job. Then, for F, new job is created and G is done in the reducer of the second job. In short, two map-reduce jobs are produced for the pigLatin program.