Tutorial 10 — Parallelism and Distributed Databases

Richard Wong
rk2wong@edu.uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

April 2, 2018

Contacting Me

If you have a question that you are okay with sharing publicly prior to the final exam, visit `https://github.com/rwongone/ece356/issues` and open a new issue.

You can still email me at `rk2wong@edu.uwaterloo.ca`, but for the benefit of your peers I may ask for your permission to share your question and my answer via the Issues page.

Exercise 10-1

What kinds of queries are the following partitioning schemes well-suited for?

1. round-robin
2. range partitioning
3. hash partitioning

Exercise 10-1 Solution

1. round-robin: good for high-selectivity queries; tuples of a single relation are spread out evenly throughout the n disks, so load is also balanced. not as good for point queries and range queries, since we can't know ahead of time which disk the tuples reside on.
2. range partitioning: good for point and range queries on the partitioning attribute, since we can quickly find out the disk or range of disks that the tuples reside on.
3. hash partitioning: good for point queries on the partitioning attribute. Also good for high-selectivity queries since tuples are evenly distributed.

Exercise 10-2

How would a distributed DB using the following partitioning schemes handle **addition** of nodes?

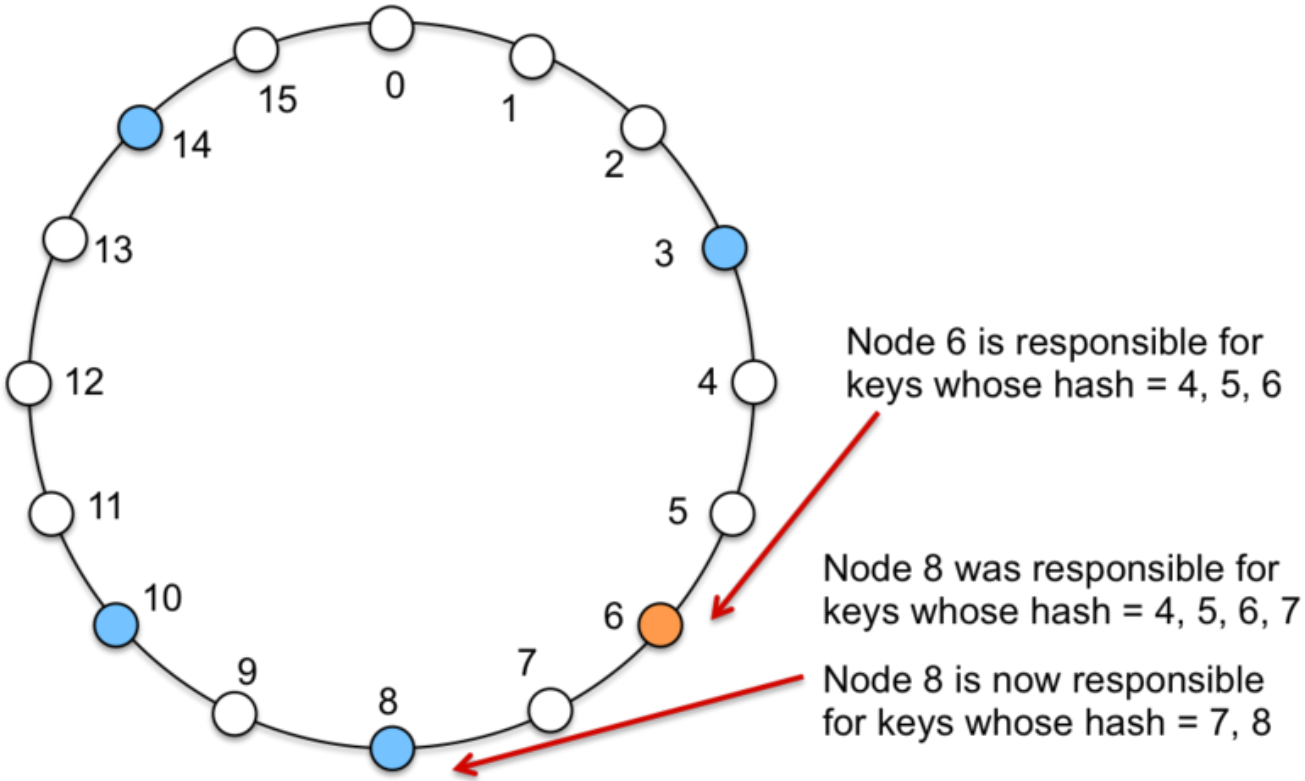
1. round-robin
2. range partitioning
3. hash partitioning

Exercise 10-2 Solution (1/2)

1. round-robin: need to recompute which partition each tuple will go to. Recall the partition is decided by some counter mod n, but by adding a new node, n changes.
2. range partitioning: the new node is responsible for a new range, and takes data from the nodes responsible for the ranges on its left and right.
3. hash partitioning: if we implement this naively, (i.e. partition is $h(x)$ mod n), then do the same as in round robin; not great.

Exercise 10-2 Solution (2/2)

Note: we can use a "distributed hash table" to make node addition with hash partitioning more like it is with range partitioning. One implementation is called Chord.
tldr: "Hash the tuples, but also hash the nodes. Each node is responsible for the range of hashes to its right."



Credit for image: https://www.cs.rutgers.edu/ pxk/417/notes/23-lookup.html

Exercise 10-3

Distinguish between the following:

1. attribute-value skew
2. partition skew
3. execution skew

Exercise 10-3 Solution

1. attribute-value skew: many tuples with the same attribute or packed in a tight range of attributes reside on a single partition.
2. partition skew: partitions carry uneven quantities of tuples, independently of attribute-value skew. This seems to be an effect of random chance, or a poor partitioning scheme.
3. execution skew: during the execution of a query, a few disks may contain most of the tuples to access, while other have very few or none. This creates an I/O bottleneck, and is also known as *load imbalance*.

Exercise 10-4

What factors would account for **execution skew** in the following partitioning schemes?

1. range partitioning
2. hash partitioning

Exercise 10-4 Solution

Attribute-value skew tends to lead to execution skew, in both range- and hash-partitioning.

A naive implementation of range partitioning will allocate most of the load to a small number of disks (just the ones that cover the query range).

Exercise 10-5

How can we alleviate the problem of load imbalance in range partitioning?

Exercise 10-5 Solution

One strategy is to use **virtual processors**.

Pretend that there are many more virtual processors as there are real processors.

Instead of mapping real processors to attribute ranges, we map the many virtual processors to attribute ranges, then balance the virtual processors among the real processors.

This causes our ranges to be split more finely, and also distributed evenly among the disks, causing range queries to hit a greater number of disks, which allows for greater parallelism.

Exercise 10-6

Suppose we have the following relation:

```
employee(name, address, salary, plantNumber)
```

The relation is **fragmented horizontally** by `plantNumber`,
and each fragment has a **local replica**,
and a **replica in New York**.

Provide a reasonable processing strategy for each of the following queries **made from the plant in Montreal**:

1. Find all employees at the Toronto plant.
2. Find the average salary of all employees.
3. Find the highest-paid employee in Toronto, Vancouver, and Edmonton.

Exercise 10-6 Solution

```
employee(name, address, salary, plantNumber)
```

1. Find all employees at the Toronto plant:
   1. send the query to Toronto
   2. process in Toronto
   3. receive the result from Toronto
2. Find the average salary of all employees:
   1. send the query to New York
   2. compute the average in New York, since it has a replica of all fragments
   3. receive the result from New York
3. Find the highest-paid employee in Toronto, Vancouver, and Edmonton:
   1. send query to Toronto, Vancouver, and Edmonton
   2. compute max at those sites to take advantage of parallelism
   3. receive results Toronto, Vancouver, and Edmonton
   4. aggregate in Montreal