Gurjus Singh

MSDS 432 Foundations of Data Engineering

August 2nd, 2020

Module 6– Reading Comprehension

1. **Describe some ways to ensure balanced partitions. (2 pt)**

   Some ways to ensure balanced partitions are creating a range of keys that correspond to a partition. This means that the keys can be located on that partition. The keys will be sorted in this method. In this idea it is also important to know the appropriate node where the partition is assigned. One negative of this way to ensure balanced partitions is that the ranges are not evenly spaced. This can lead to hotspots or overloading such as in the case a key is a timestamp as one partition will be created per day. The ranges can be chosen by an administrator or automatic.

   Another way to balance by partitions is by hashing. Some hash functions in programming languages may not be best for hashing because they might have different hash codes for the same key in different processes. The best thing about hashing it makes a skewed distribution of data storing more uniform. The bad side of hashing is we are unclear as to where the data resides as keys may reside an unknown location which become inefficient when finding the data when querying.

   Hashing can also be problematic by avoiding hot spot reduction such as if you have two of the same keys, they can both lead to the same partition. A workaround for this is to add a random number to the front or end of a key. This will also create a downfall as it will take longer to query.

2. **Describe the two main approaches to partitioning a database with secondary indexes. (2 pt)**

   One main approach for partitioning a database is with secondary indexes by document. This works by each document having a unique ID called a Document ID. Then the database can partition the database by the Document ID. The way it will be partitioned is by ranges of ID. Once they are partitioned by index, then in order to filter terms, a secondary index is created based on the terms, and a list of documents is created that contain the filtered term.

   Another main approach to partition a database with a secondary index is by term. With this approach there is a global index. For example, if the documents are on different partitions, the IDs can be combined in a list from all the partitions and be indexed according to alphabet letter ranges where the ranges are split between partitions. For example, if the index is a list of colors a-c colors will be on one partition, where all the documents that reference that color in that range are on that partition.

D-f color range are on another partition and so on and the same approach applies. The main advantage is this can make queries faster. The database already does the collecting of the documents that contain a specific phrase. Writing to a database may be more tedious because you have to keep updating the index. Also when updating you have to keep track of references when the document IDs partition changes when rebalancing. Also secondary indexes can be asynchronous so it takes time to update when the data is updated.

3. **Describe some ways to rebalance partitions. (2 pt)**

   Some ways to rebalance partitions are to use a fixed number of partitions. The solution if you use this method is to use more partitions than nodes. For example, you can assign 1,000 partitions to 10 nodes. This is creates 10,000 partitions. If a new node is added, then it will steal some partitions from other nodes to make it distributed. The reverse can happen when a node is reversed. If there are nodes that are more powerful in comparison to other nodes it will be better to assign more partitions to this powerful node.

   Another way to ensure rebalance partitions is through dynamic partitioning. The way this works is by adapting the number of partitions to the data size. Databases such as HBase and Rethink DB use this because they are key range partitioned. The way they use dynamic partitioning is when a partition gets bigger to a set threshold, then the partition gets split into two partitions. If the database shrinks, then again the partitions merge with a close by partition. When the database is empty, there is only one partition on a single node until the node reaches a threshold till other nodes are involved.

   Another way to rebalance partitions is by partitioning proportionately to the nodes. This happens when there are unchanging number of partitions on one node. When the partitions are fixed the size of each partition will grow proportionately to the data set size. When the nodes are increased, then the partitions will become smaller again.

4. **Describe the concept of service discovery and some ways of dealing with it. What are some existing products that deal with this concept in terms of distributed data systems? (2 pt)**

The concept of service discovery involves when a client wants to make a request for data. How can they know which node the data resides on such as the IP Address and port number? This is not just limited to databases but can happen on any software over the internet or network. Another problem we are concerned is how do nodes make the routing decisions when making requests from the client? How do they know about changes on other nodes?

There are processes that can help deal with this concept, for example, clients can contact any node, and the node can deal with it directly or forward it and then receive the reply back to pass back to client. Another process is to use what is known as a routing tier to determine which node can be used so this can handle the request without making the load to high. Thirdly, the client can also be aware of processes of partitioning to keep track of where to send requests, or it can do it on its own.

Specific tools to use for this concept are Zookeeper to keep track of cluster information. The Nodes relay information to Zookeeper. This also involves partition ownership and node location. The Zookeeper can then notify the routing tier for load balancing purposes. Databases such as Cassandra and Risk use gossip protocol to spread any changes in cluster state.

5. **(Algorithms) What are the different parts of a graph and how do they affect the complexity (in terms of O notation) of the Breadth First Search algorithm? (1 pt)**

   A graph is a model of a set or connections. It consists of nodes and edges. For example, nodes can be people, while edges can be a friendship between two people. This exactly how social media sites like Facebook work. A node can also have one to many relationships such as a person who has a bunch of friends, and edges are the many relationships. With this information it is important to note the running time of Breadth First Search is O(V + E) where V are the nodes or known as vertices while E are the number of edges. This is the running time because you are essentially searching each person to find the correct person, but you are also going through each person's connections which is where the Edges come into the picture.

6. **(Algorithms) What is the difference between a Breadth First Search and a Depth First Search? When should someone use one over the other? (1 pt)**

   Depth First Search and Breadth First Search are two search algorithms. The difference is the data structures they use such as a stack for DFS and while BFS uses a queue. For BFS it goes level by level by going closest to the starting node and moving on to the next level farthest away, while DFS is searches the starting node, then searches out every iteration. BFS would be used when you are more worried about finding out the shortest path, while DFS is more of a recursive algorithm. Both of the algorithms have the same run time which is O(V + E). Depth First Search uses less memory than BFS.