6. Partitioning

- Partition
  - aka shard in MongoDb, aka region in Hbase, aka tablet in BigTable aka vnode in Cassandra/Riak, aka vBucket in Couchbase
- Partitioning and Replication
  - Replication of partitions - is a common practice
- Partitioning of Key-Value Data
  - 👎 Skew
    - When partitioning is unfair, so that some partitions have more data or queries than others
  - 👎 Hot spot
    - Partition with disproportionately highload
  - Partitioning by Key Range
    - 👍 Efficient for range query
    - 👎 Some access patterns can lead to hot spots
      - e.g. partitioning by datetime - all writes will target the same partition
      - Possible solution
        - Prefix with sensor name
  - Partitioning by Hash of Key
    - 👍 Reduce risk of hot spot
    - approaches
      - Ranges of hashes
      - Consistent Hashing
        - Pseudorandomly assign to partition
    - 👎 Can't do efficient range query
      - Possible solution - hybrid approach
        - concatenated index approach
          - First compound column for partitioning (user_id, update_time-stamp), others for range scan (update_timestamp, ...)
  - Skewed Workloads and Relieving Hot Spots
    - Sometimes we still have hot spot
      - E.g. lots of comments on some popular post
    - 👍 Possible solution
      - Add random number
        - But only to really HOT records not for all!
- Partitioning and Secondary Indexes
  - Partitioning Secondary Indexes by Document
    - local index
      - 👍 Each partition has secondary indexes with only data in this partition
      - 👎 Select need to query all partitions - scatter/ gather
        - prone to tail latency amplification
  - Partitioning Secondary Indexes by Term
    - global index
      - Covers data in all partitions
      - Index should be partitioned itself - term-partitioned
        - Partitioning by term
          - Good for range scans
        - Partitioning by hash of term
          - More even distribution of load
      - 👍 Client can read from only one right partition
      - 👎 Writes are slower and more complicated
        - Write to single document may cause writes to many partitions (to write in many indexes)
        - In practice this is often async
- Rebalancing Partitions
  - Strategies for Rebalancing
    - 👎 How not to do it: hash mod N
      - If N changes most of the keys need to be mvoved to another partition
    - Fixed number of partitions
      - We have 1000 partitions and 10 nodes
      - 👍 The only thing that changes is the assignment of partitions to nodes
      - 👍 Can even assign more partitions to more powerful nodes!
      - 👎 Hard to chose right number of partitions if data grows very fast
    - Dynamic partitioning
      - When a lot of data - divide, when too few data - merge
        - similar to what happens at the top level of a B-tree
      - 👍 Number of partitions adapts to the total data volume
      - 👎 Starts from single partition - so not efficient when too few data
        - Solution
          - 👍 pre-splitting
            - Manual initial settings
      - Supports both
        - Range-partitioned
        - Hash-partitioned
      - number of partitions is proportional to the SIZE OF DATASET
        - vs
    - Partitioning proportionally to nodes
      - number of partitions is proportional to the NUMBER OF NODES
        - i.e. fixed number of partitions PER NODE
  - Operations: Automatic or Manual Rebalancing
    - Fully automated
      - 👍 Convenient
      - 👎 Unpredictable
        - It is expensive operation
        - Can cause cascade failure
          - When node is overloaded and system decides to rebalance data from it - and it increases load on it!
    - So - at least partially manual is better
- Request Routing
  - service discovery
  - Few approaches
    - Send request to random node
      - round-robin load balancer
        - Redirects to next node if this node is not owner of required partition
    - Send all requests to routing tier first
      - partition-aware load balancer
    - Require client aware of the partitioning
  - Key issue
    - ⚠️ How decision maker should learn about changes in the assignment of partitions to nodes?
      - Use protocols to achieve consensys
        - But they are hard to implement
      - Use coordination service
        - e.g. Zookeeper
        - Everybody subscribes to it's changes
  - Parallel Query Execution