PES UNIVERSITY

CLOUD COMPUTING

Rebalancing Partitions and Request Routing

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:
Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Skewed Workloads and Relieving Hot Spots

- We saw that Hashing a key to determine its partition can help reduce hot spots. However, it can’t avoid them entirely
- In the extreme case where all reads and writes are for the same key, all requests will be routed to the same partition.
- Most data systems may not be able to automatically compensate for such a highly skewed workload, and this may be handled by simple techniques like adding a random number to the beginning or end of the key in case of one of the key is known to be very hot or the applications may need to support this. This can address it in a limited fashion and can cause lower performances as for reads data will need to be read from all the keys and combined.
- Distributed hashing discussed earlier, will help in distributing and supporting the throughput, but there will always be scenarios where a workload may need to be moved across nodes.

Rebalancing Partitions

- Since over time, things change like
  - The query throughput in a Database increases, so you want to add more CPUs to handle the load.
  - The dataset size increases, so you want to add a new partition with appropriate capacity
  - A machine fails, and other machines need to take over the failed machine's responsibilities.
  - A Machine joins in .. A new consumer or an recovered machine
- All of these changes call for data and requests to be moved from one node to another. The process of moving load from one node in the cluster to another is called **rebalancing**.
- Rebalancing is expected to meet some minimum requirements regardless of the partitioning scheme:
  - After rebalancing, the load (data storage, read and write requests) should be shared fairly between the nodes in the cluster.
  - While rebalancing is in progress, the database should continue to accept reads and writes.
  - No more data than necessary should be moved between nodes, to make rebalancing fast and to minimize the network and disk I/O load.

https://notes.shichao.io/dda/ch6/#:~:text=Document%2Dpartitioned%20indexes%20(local%20indexes,scatter%2Fgather%20across%20all%20partitions

Strategies for Rebalancing Partitions: Hash based rebalancing : hash mod N

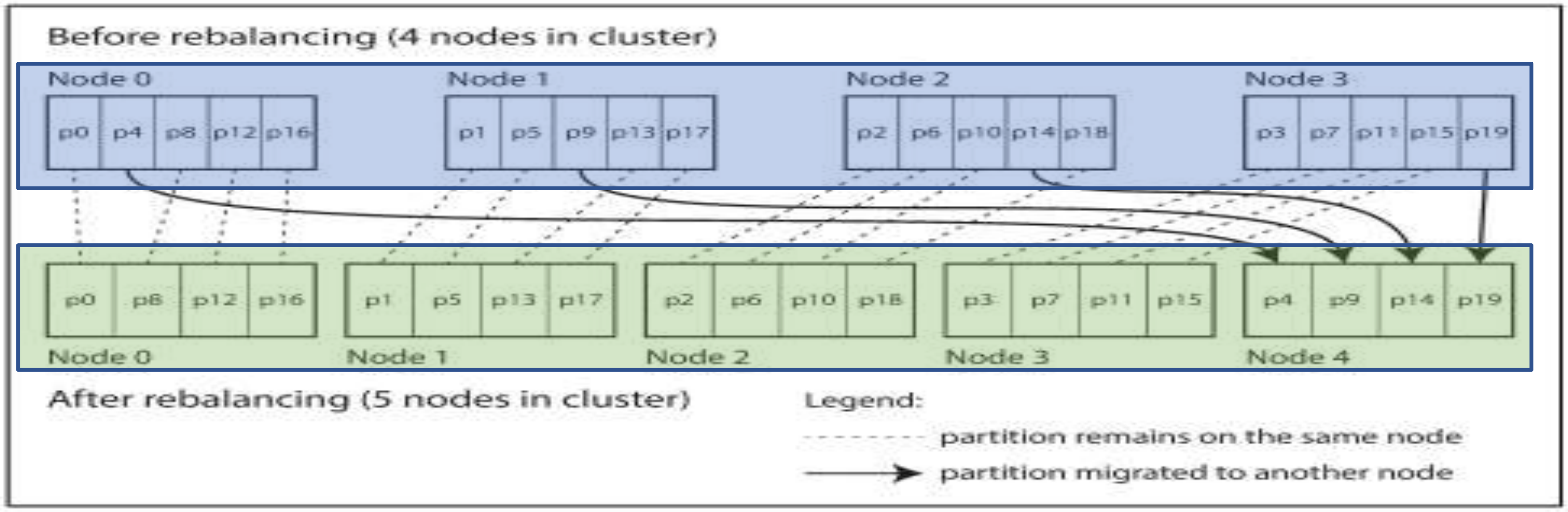
- We had discussed as part of distributed hashing, an approach of distribution of the keys onto different servers using a simple hash modulo of the number of servers in the environment
- So, we partition by the hash of a key, and divide the possible hashes into ranges and assign each range to a partition (e.g., assign key to partition 0 if $0 \le hash(key) < b0$, to partition 1 if $b0 \le hash(key) < b1$ etc.) and just use mod (the % operator in many programming languages) to associate a partition to a node.

  E.g. Say if we have 10 nodes (numbered from 0 to 9), hash(key) mod 10 would return a number between 0 and 9 (if we write the hash as a decimal number, the hash mod 10 would be the last digit) which seems like an easy way of assigning each key to a node.
- We also discussed that in this approach if the number of nodes N changes, most of the keys will need to be moved from one node to another.
- So this frequent moves make rebalancing using this approach to be excessively expensive.

Strategies for Rebalancing Partitions : Fixed number of partitions

- Create many more partitions than there are nodes and assign several partitions to each node
- If a node is added to the cluster, the new node can steal a few partitions from every existing node until partitions are fairly distributed once again

Eg. If there are 04 nodes and 20 partitions. And we add an additional node.



Strategies for Rebalancing Partitions : Fixed number of partitions (Cont.)

- Only entire partitions are moved between nodes.
- The number of partitions does not change, nor does the assignment of keys to partitions. The only thing that changes is the assignment of partitions to nodes
- This change of assignment is not immediate. It takes some time to transfer a large amount of data over the network so the old assignment of partitions is used for any reads and writes that happen while the transfer is in progress.
- Choosing the right number of partitions is difficult if the total size of the dataset is highly variable
- The best performance is achieved when the size of partitions is “just right,” neither too big nor too small, which can be hard to achieve if the number of partitions is fixed but the dataset size varies.
- A fixed number of partitions is operationally simpler. So many fixed-partition databases choose not to implement partition splitting but use fixed number of partitions

Strategies for Rebalancing Partitions : Dynamic Partitioning

- For databases that use key range partitioning , a fixed number of partitions with fixed boundaries would be very inconvenient if the boundaries are wrong. All of the data could end up in one partition and all of the other partitions could remain empty
- Alternatively if the partitions can dynamically be created, say when a partition grows to exceed a configured size , it is split into two partitions so that approximately half of the data ends up on each side of the split. One of its two halves can be transferred to another node in order to balance the load.

  If lots of data is deleted & a partition shrinks below some threshold, it can be merged with an adjacent partition
- Advantage of this dynamic partitioning is that the number of partitions adapts to the total data volume
- Each partition is assigned to one node and each node can handle multiple partitions
- In this approach if we start of with a single partition (say like an empty DB), all writes have to be processed by a single node while the other nodes sit idle.
- Dynamic partitioning is not only suitable for key range partitioned data, but can equally well be used with hash partitioned data

Strategies for Rebalancing Partitions : Partitioning proportionally to nodes

- With *dynamic partitioning, the number of partitions is proportional to the size of the dataset*, since the splitting and merging processes keep the size of each partition between some fixed minimum and maximum
- With a *fixed number of partitions*, the *size of each partition is proportional to the size of the dataset*.
- In both the above cases, the *number of partitions* is *independent of the number of nodes*
- Make the *number of partitions proportional to the number of nodes*—in other words, to have a *fixed number of partitions per node*
- The *size of each partition grows proportionally to the dataset size* while the *number of nodes remains unchanged*, but when you increase the number of nodes, the partitions become smaller again
- Since a larger data volume generally requires a larger number of nodes to store, this approach also keeps the size of each partition fairly stable.

Strategies for Rebalancing Partitions : Partitioning proportionally to nodes

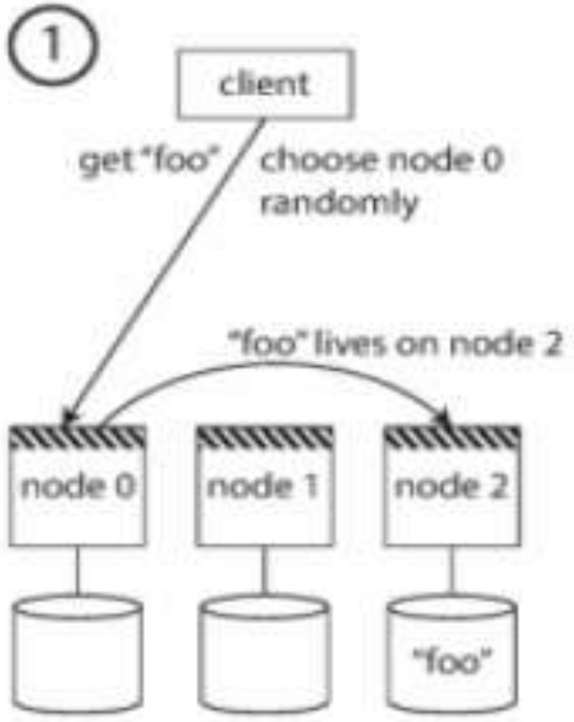
- When a new node joins the cluster, it randomly chooses a fixed number of existing partitions to split, and then takes ownership of one half of each of those split partitions while leaving the other half of each partition in place
- The randomization can produce unfair splits, but when averaged over a larger number of partitions, the new node ends up taking a fair share of the load from the existing nodes.
- Picking partition boundaries randomly requires that hash-based partitioning is used

CLOUD COMPUTING
Request Routing

- Consider a partitioned data distributed across multiple nodes running on multiple machines
- If a client wants to make a request, how does it know which node to connect to?
- As partitions are rebalanced, the assignment of partitions to nodes change. So there is a need to stay on top of those changes in order to provide information like which IP address and port number to connect to.
- This is typically called the service discovery problem which has been addressed with approaches as below.

Approach 1

Allow clients to contact any node (e.g., via **a round-robin load balancer**). If that node coincidentally owns the partition to which the request applies, it can handle the request directly; otherwise, it forwards the request to the appropriate node, receives the reply and passes the reply along to the client.

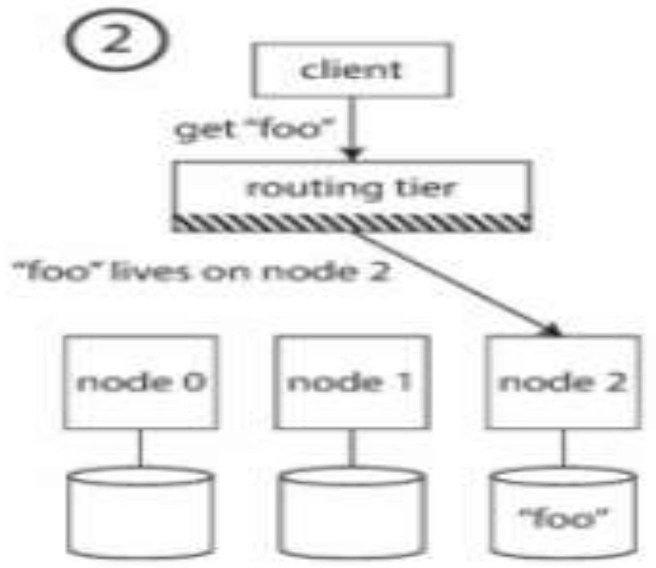


CLOUD COMPUTING

Request Routing

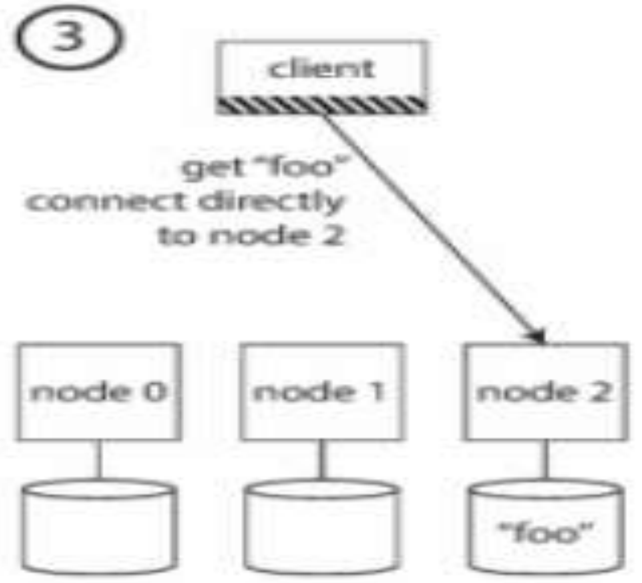
Approach 2

Send all requests from clients to a routing tier first, which determines the node that should handle each request and forwards it accordingly. This routing tier does not itself handle any requests; it only acts as a **partition-aware load balancer**.



Approach 3

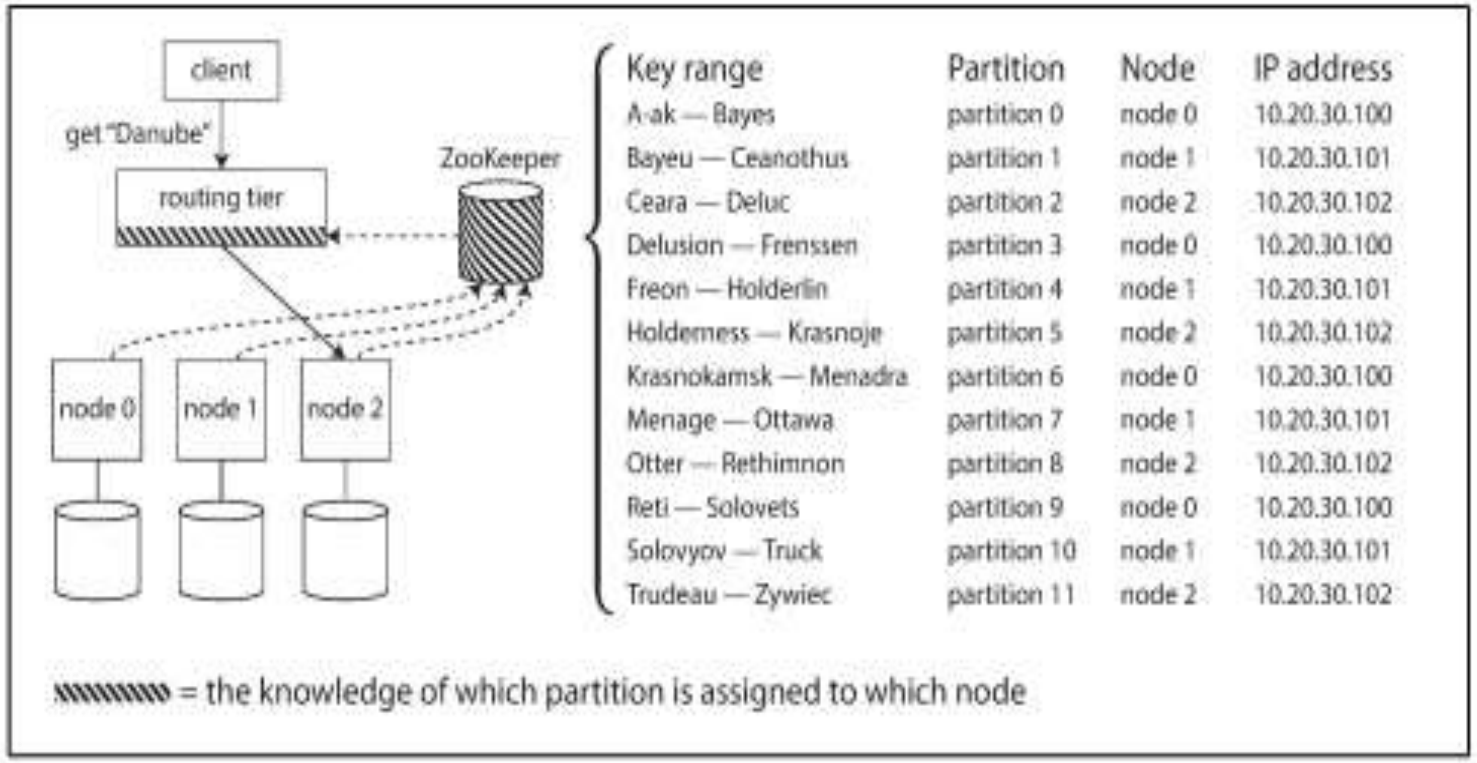
Require that clients be aware of the partitioning and the assignment of partitions to nodes. In this case, a client can connect directly to the appropriate node, without any intermediary



Request Routing

Approach 4 : ZooKeeper

- Many distributed data systems rely on a separate coordination service such as ZooKeeper to keep track of this cluster metadata
- Each node registers itself in ZooKeeper, which maintains the authoritative mapping of partitions to nodes.
- Other actors, such as the routing tier or the partitioning-aware client, can subscribe to this information in ZooKeeper.
- Whenever a partition changes ownership or a node is added or removed, ZooKeeper notifies the routing tier so that it can keep its routing information up to date.



Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

▨ = the knowledge of which partition is assigned to which node

ZooKeeper - A Distributed Coordination Service for Distributed Applications

https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index

- HBase, SolrCloud and Kafka use ZooKeeper to track partition assignment
- **(Revised versions of Kafka have discontinued ZooKeeper and this logic is replaced with KRAFT)**
- LinkedIn's Espresso uses Helix for cluster management (which in turn relies on ZooKeeper), implementing a routing tier
- Cassandra and Riak use a gossip protocol among the nodes to disseminate any changes in cluster state. Requests can be sent to any node, and that node forwards them to the appropriate node for the requested partition

(https://medium.com/@roopa.kushtagi/understanding-the-evolution-of-zookeeper-in-apache-kafka-eb8dbeed460f#:~:text=Beginning%20with%20Kafka%20version%202.8,improved%20performance%2C%20and%20simplified%20operations.)

THANK YOU

Prafullata Kiran Auradkar
Department of Computer Science and Engineering
prafullatak@pes.edu