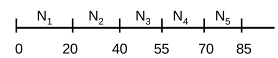
Online Balancing of Range-Partitioned Data with Applications to P2P Systems

Prasanna Ganesan
Mayank Bawa
Hector Garcia-Molina

Motivation

- Support range queries in a P2P system
  - SELECT * FROM Auctions WHERE price < 1
- DHTs inefficient for range queries
  - Useless if range is not enumerable
- Solution: Range partitions instead of hash



The Problem

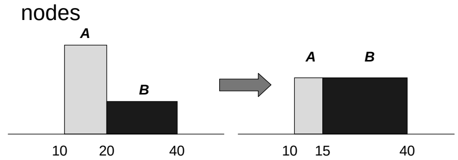
- How to achieve load balance?
  - Nodes join/leave, Data is inserted/deleted
  - Partition boundaries have to change over time
  - Cost of achieving balance: data movement
  - Goal: Guarantee load balance at low cost
- More motivation for not using hashing
  - Hashing offers balance only for unique keys
  - Key distribution often Zipfian

A Parallel Database Setting

- Assume fixed set of nodes
  - Tuples are inserted/deleted over time
- Nodes always maintain a range partition
  - Allowed to modify the partitions
  - Each tuple transfer between two nodes costs 1 unit
- Load imbalance σ = Largest load/Smallest load
- Goal: Ensure σ < *constant* with *constant* cost per tuple insert/delete

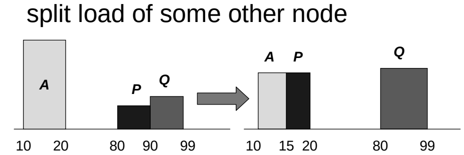
Load Balancing Operations (1)

- NbrAdjust: Transfer data between adjacent nodes



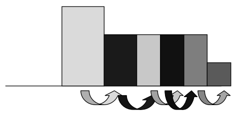
Load Balancing Operations (2)

- Reorder: Hand over data to neighbor and split load of some other node



Why two operations?

- Can balance load with just NbrAdjust
  - $\Omega(n)$ amortized cost per insert/delete in worst case: Inefficient



- Need Reorder for efficiency
  - Both necessary and sufficient

The Doubling Algorithm

- Each node monitors its own load
- If load crosses threshold, try balancing
  - Threshold=all powers of two
- When load increases beyond threshold 2^x
  - If $L(nbr) \le 2^{x-1}$, do NbrAdjust to equalize
  - Else if $L(lightest\text{-}loaded) \le 2^{x-2}$, do Reorder with *lightest-loaded* and its neighbor
  - Similar operations when load decreases below threshold
  - Some additional recursive balancing needed

Costs and Guarantees

- Load imbalance $\sigma \le 8$ always
  - Nobody larger than 2^x if someone smaller than 2^{x-3}
- Cost per insert/delete of tuple?
  - Guaranteed to be amortized constant
  - i.e., cost after *m* operations $\le cm$
- Note similarity to DHT guarantees

Improving on Doubling

- Change thresholds to Fibonacci numbers
  - $\sigma \le$ cube of golden ratio
  - Can also use other geometric sequences
  - Costs are still constant
- Dealing with concurrent inserts/deletes
  - Allow multiple balancing actions in parallel
  - Paper claims it is ok

Implementing in a P2P System

- Support required for load balancing
  - Deal with node joins and leaves
  - How to find lightest-loaded node?
- Support required for queries
  - Need efficient routing of range queries, *a la DHTs*

Node joins and leaves

- Node join
  - Split heaviest-loaded node!
- Node leave
  - If no data replication, easy to re-balance
  - If data is replicated, tuples are “re-inserted”
- Costs
  - Proportional to average load in system
  - Note: Cannot do better

Handling Queries

- Create a skip graph
  - Linked list of nodes, ordered by ranges
  - Plus, *O(log n) skip pointers* per node
  - Performance similar to Chord, etc.
  - Note: Plain Chord with node ID=left end of range does not work
- Range query for data in [a,b]
  - Route to reach node containing a
  - Traverse linked list from thereon
  - *O(log n + fn)* messages; *f=query selectivity*

Finding lightest-loaded node

- Build another skip graph!
  - Arrange nodes according to load
  - Update whenever loads cause change in ordering
  - Can be done with *O(log n)* messages per insert/delete
- Or, simply use sampling
  - Look at a small number of loads and take the lightest seen
  - Shown to work well in practice

Experiments

- Does algorithm beat periodic reorganization?
  - Yes, by a giant factor for some workloads
- What are the costs per insert/delete?
  - For static data distribution, goes to zero
  - For "adversarial" workloads, still under 2
- Do load balance guarantees hold?
  - Yes. Usually better than the guaranteed value.
  - Worsens with use of randomization

Conclusions

- Indeed possible to support range queries in P2P systems
  - Same asymptotic costs as DHTs for tuple insert/delete/point query
  - Same/better load balance as DHTs
  - Range query requires additional *O(fn)* messages, which is optimal given load balance
- Algorithms useful for *dynamic* load also
  - Change load definition in algorithms!
  - Need load to be "evenly divisible"
- Also important when search key is not unique
  - E.g., inverted index of keywords

Alternatives to Range Partitions

- Prefix tree of data blocks. Store in DHT?
  - Data-dependent query costs
  - Fragmentation of data across nodes
- Build a B-tree and distribute blocks by hashing
  - Too much meta-data to update on node failure
  - Need to replicate root block, etc. for routing load balance