DP-Trees: A Distributed Ranged Index
Sarah Mitchell and Sean Neubert

1. Introduction

P2PTrees are a distributed index structure designed for fast searches among a number of computers that can communicate across a network. When working with a single value per peer or with multiple values that are clustered, this is an effective solution. However, when working with a distributed system in which a peer is likely to have values that are divided along a range, the P2PTrees often have to use an average value when responding to queries. This can diminish the efficiency and accuracy of searching. P2PTrees are insufficient for managing peers on a network in which multiple records that aren’t clustered are stored in each peer.

The solution is to distribute a ranged index across multiple peers. Each peer will store two values, a high value and a low value. For ranged data, this can be two points that encompass the region it covers, much like an R-Tree. In this respect, a ranged query can effectively be performed. This index structure implements searching in two phases. The first phase is a search performed using this index structure. This searching is only done to determine a set of peers that could contain the value searched for. The second phase of searching is to search each of the peers within this set. Sorting conditions are applied to each peer, so the results of each one is ordered. This allows the client to implement a merge sort of the data returned.

Any Peer in the network is responsible for maintaining a tree containing any peer that is entirely or partially inside of its range of values. The easiest way to think of this is as a map. If you look at a map of the world, it contains many nations. Many nations have states or provinces, and some of them contain counties. That’s the way that this structure works. A peer contains its own range of values or points, and any other peer that has a range that overlaps with it or that has a range entirely within it is stored.

It is important to note that each of these trees stores a root and peers below it. In the tree, we don’t worry about storing the children of a peer, we only worry about storing peers that are in our range or that overlap. If we want to examine the children of another peer, we contact it. This reduces storage and communication needs when adding and removing peers.

This new index structure is called a DP-Tree, and the “D” stands for “Distributed”. No index structure lacks any drawbacks, and DP-Trees have them as well. This index is designed for the purpose of a small to moderate number of dedicated machines to provide access to data. It is not designed for a peer-to-peer network, in which peers are added and removed very frequently. Also, the method in which data is distributed greatly affects performance. If each peer contains the same range, than performance will be worst case. The best performance occurs when each peer has a distinct range.
2. Design and Architecture of DP-Trees

2.1 Operators:

**Comparisons:** The issue with ranged peers becomes ordering. How does ordering work when each peer has two values or points? The comparison is similar to a P2P-Tree, in that comparisons are done within the context of the current peers range. Any peer with a high value greater than the current peer is a larger value. Any peer with a low value lower than the current peer is a lower value. As an example, consider the following peers than the ordering: Peer A (2, 10), Peer B (5, 7), Peer C (19, 24), Peer D (29, 34), Peer E (2, 15), and Peer F (4, 13). Let’s look at ordering with regard to Peer B. Peers A, E, and F are all less than B. Peers C and D are greater than B. The order of least to greatest is: A, E, F, B, C, and D. A is less than E although they have the same low value, that is because the value with the larger range is considered closer.

2.2 The data structure:

**A Peer:** A peer, when stored within a tree, represents a range of values that another peer contains.

**A Node:** Nodes are stored in a tree when a level in the tree overflows (too many peers are stored). When this occurs, Nodes are created that then store the peers. A Node is like a peer in that it stores a range, but it is only used for

**The Tree:** The tree stores anywhere from two to n levels. The first level is the root, which contains the current peers range of values. The second level contains nodes or peers. When the first level reaches its size limitation,

**The ring list:** Each peer contains a previous and a next peer, so that it can keep track of the two peers that are in either direction of the peer should a request be outside of its range. However, if the previous or next peer fails, than the peer needs to know

**Central Server(s):** The central server(s) are an optional feature and are run as a means of keeping the distributed index in order. The central servers are kept up to date on what peers are added and removed from the network and are available if a peer has difficulty with operating. It is also a central location for error reporting. If no central server is used, than every peer maintains a list of all of the other peers that exist on the network. This list isn’t used for searching, but for recovery during peer failure and when a new peer joins the network.

3. Applications

A database distributed across many networked computers could use this distributed index structure. Because this design can accompany single and multiple dimensions, an R-Tree or a B-Tree variant can be replaced with this structure when the data is distributed. Distributed databases, whether because of size limitations,
redundancy, or for a greater response time for a database server.

4. Algorithms

4.1 Query Algorithms

Value Searching:
When a peer is queried with a value, there are three possible courses of action. If the search value is within the range of the current peer, than the current peer searches it for any other peers that might contains its value. Those peers are contacted and searched for any other peers that fall within this range. The results are sent back to where the request originated, so that peer can prevent duplicates from entering the search list. Otherwise, an infinite loop could occur.

Range Searching:
A range query is no different from a value query, except that when searching the range is examined to see if the query range and the peer range overlap. If they do, the peer is added to the result list, otherwise it is not.

Inserts:
When a record is to be inserted, a search is performed to find out what peers have ranges that can accommodate this value. The average value of each peer is computed, and the peer whose average value is the shortest distance to the new value is given the new record to insert. If no peer has the range, the top-level peers are all searched in order to determine which one would require the lowest amount of expansion. That peer then expands its range and adds in the new record.

Removals:
Removals are quiet easy. A search is performed, and each of the returned peers is queried to find the value. When it is found it is removed from the peer that contains it.

4.2 Operational Algorithms

Adding a peer:
When a peer is added to a network, the simplest possible case is when it is the first node, in which case it is added to the tree as the one and only node. If a peer is added to a network with other peers, than the new peer is given a list of all of the other peers. Suppose a network exists with a single peer labeled A. Then a peer labeled B joins the network. A provides B with a list of all current nodes in the network, which is only A. B than uses A to modify its tree. If the range of A's values overlap with B, than B adds A to itself. A does likewise. If the ranges do not overlap, than the other peer is not added to the tree.

Modifying data such that the range of a peer changes:
If a central server is used, than a message is sent to it and the central server than relays the message to all other peers. If no central server is used, than the peer itself relays this information to all other peers.
**Peer Failure:**
A peer failure can be found based upon two different circumstances. The first is when the ping process cannot communicate with a node. This will almost always mean that the peer is unreachable over the network or that the peer has crashed, and so that is what is assumed. The other case is when one peer tries to contact another for an insert, a query, or delete operation and it cannot be reached.
To handle this, a data structure is stored in each peer that keeps track of the next $n$ and previous $n$ peers in the ring. Should a peer try to contact its next or previous peer and fail, it will look in the list to find the next one to contact. If all of these peers are unable to be reached, than the peer will assume that it is the peer that is failing and cease operations. This will not happen, however, if central servers are specified. In this case, the peer will contact one of the central servers and request the next and previous values it should contain.

**Conclusion:**
DP-Trees provide a way of storing a range of values for effectively searching and managing both single and multidimensional data across multiple computers.