Cluster-Computing and Parallelisation for the Multi-Dimensional PH-Index

Master Thesis

Bogdan Aurel Vancea
<bvancea@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Tilmann Zaeschke
Christoph Zimmerli

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zurich

10th February 2015

Abstract

Here comes the abstract.

Contents

1 Introduction

1.1 Multi-dimensional Indexes

1.1.1 Background information

Need to add an introduction here.

This is an example of how to cite a scientific publication [1] from your bibliography (BibTeX[1] file). And this example shows how you create links within your documents, e.g. link to section 1.1.

[1]http://en.wikipedia.org/wiki/BibTeX

2 Design and Implementation

2.1 Algorithms

2.1.1 Requirements

The distributed index has the following requirements:

1. Eventual consistency. The index is eventual consistent, meaning that the following assumptions are true:
   (a) Any entries that are inserted at some point t in time will eventually be returned in results at a following point in time t + ϵ .
   (b) In the case that a client starts an insert operation at time t and another client, in a separate thread or a separate, issues a query at time t + ϵ, the response of the query of the second client is not guaranteed to contain the point inserted by the first client.
2. Re-balancing transparency. The balancing process should be transparent with respect to the client. This means that the client API should not know that there might be multiple versions of the key mapping available due to any on-going balances.

2.1.2 Data partitioning

Describe here how the key-value pairs are partitioned across the hosts.

2.1.3 Data balancing

Describe how the key-value pairs are balanced across the index nodes. The cluster should be able to properly balance the amount of keys that are stored in each server.

Ideas that might not work:

- Storing information like the number of keys held by each node in Zookeeper. This would mean that the ZooKeeper would need to be notified on each insert, which would cause severe scalability and availability issues.

The size information linked to each host is very important for properly spreading the entries across all of the hosts. To prevent the ZooKeeper server from becoming a bottleneck while updating the sizes, the update of the size information should be sent periodically, but not after each insert / deletion operation. A possible solution is for each host to update it's size after a fixed number of insert/delete operation. Another possible solution is to have each host update the size after a fixed period of time, like 1s, 10s, etc.

2.1.4 Basic Index operation

Describe the point operations here. These are operations that affect a single key-value pair, like get, put, delete, contains, etc.

2.1.5 Iterators

Describe how iterators are handle when dealing with a cluster of index servers. Describe the current algorithm used and the alternatives.

2.1.6 Range search

Describe how the range search is performed. Describe how the number of hows that need to be querried is reduced and what the alternatives are.

2.1.7 K Nearest neighbours

The K nearest neighbours should be found in the following manner:

1. First, a request for the k nearest neighbours is sent to the host that holds the query key. This query will return a number of candidate points.
2. After finding the furthest neighbour fn from the set of candidates obtained at the previous step on could do the following:
   - look into all neighbouring ares and check for neighbours that are closer to q than fn.

- find all zones intersecting the square (q - dist(q, fn), q + dist(q, fn)) and perform a knn query into those areas. Then apply an additional knn to combine candidates.
- find all zones intersecting the square (q - dist(q, fn), q + dist(q, fn)) and perform a range query in those areas. Then apply an additional knn to combine candidates.
- find all zones intersecting the square (q - dist(q, fn), q + dist(q, fn)) and perform a range query followed by a filtering based on dist(q, fn) in those areas. Then apply an additional knn to combine candidates.

This algorithms works under that assumption that all hosts hold at least k points. If that is not the case, and fewer candidates are returned from the first query, one would need to increase the query space and check all of the neighboring areas of the zone holding the query key. To achieve correctness even if all the cluster holds fewer than k keys, one could iteratively query more neighbours in each steps (i.e. first query the 1 - hop neighbourhood, then the 2 - hop neighbourhood) and stop after all of the hosts have been querried.

2.1.8 PhTree Concurrency

The PhTree does not currently support concurrent write operations. There are several strategies that could be employed to add concurrent writes.

Copy-On-Write COW would make the PhTree immutable. Therefore, every write operation (insert, remove, update) will create a new version of the containing the changes to be applied. The advantages of this strategy is that it allows one writer and an arbitrarily large number of readers to access the tree concurrently. The second advtange is that this method provides very high consistency guarantees, allowing range and knn queries to run for large periods of time on an older version of the tree. The disadvantages of this method is that it is designed to only allow a single writer thread access the tree in the same time and that it has a higher memory consumption as some of the nodes need to be copied on each access.

Pesimmistic Locking Hand over hand locking strategies could be used to allow multiple writers to acces the tree. Due to the fact that iterators maintain a stack of nodes, it might be needed that some of the nodes need to be copied after modification. Advantages of this approach: Integrates multiple writers. The main disadvantages are that it have lower consistency guarantees.

Optimistic Locking Optimistic locking could be used instead of hand over hand locking. This strategy would go down the tree without taking any locks to find the node to be modified. When the node is found, the parent and the node are locked and a validation check is performed to make sure the node / parent were not changed and that they are still reachable. This approach takes fewer locks when the write contention is not very high. However, it causes starvation when the lock contention is high.

The PhTree contains the following write operations:

1. Adding new key-value pair to the tree.

2. Removing a key-value pair from the tree.
3. Updating the key for a given key-value pair.

The following node modification operations are performed on the tree:

1. New nodes are added as leaves during the insert operation.
2. Some leaf nodes could be removed during the delete operation.

2.2 Implementation

Describe the technologies used, the reasons for which these technologies were chosen and any alternatives.

2.2.1 Cluster information

The configuration server is implemented using an Apache Zookeeper server.

The operations that update the mapping have to be executed atomically on Zookeeper. Zookeeper provides the guarantee that all write operations performed on a single *znode* are executed atomically. However, all update update of the mapping modify at least two znodes: the mapping znode and the version

2.2.2 Technologies and Libraries

Currently used frameworks:

ZooKeeper ZooKeeper is used for stored cluster metadata and membership. Currently, the only alternative would have been to implement such a distributed storage manager manually. Using ZooKeeper saved a lot of development time.

Netty Netty is a Java IO library and it is used to implement the server request handling component. Alternatives would have been Java NIO library.

Kryo Kryo is very fast serialization library for Java and it is used to serialize the values that have to be stored on the server. These objects need to be transformed into a represention that can be sent over the network. Kryo is faster than the Java serialization, does not require the implementation of the Serializable interface and transforms the objects into byte arrays. This should make the representation smaller than simply transforming the object to a string.

3 Performance Analysis

3.1 Benchmark

4 Conclussions

List of Figures

List of Tables

Acknowledgements

Bibliography

[1] Alfonso Murolo. Designing wordpress themes by example. Master's thesis, ETH Zurich, 2013.