what is the problem:

1. As a part of my project work I wanted to study the performance of data systems written in a

modern/ more advanced programming language such as scala

2. This was inspired by my course work during cs165 - where we wrote the main memory optimized column store in C. During this time I found that when writing in C, even the smallest things - such as size of

a arraylist is not available to the programmer.

3. While these things are not hard to do, they add to the number of things to do ( akin to the number of instruction a computer must excecute).

4. So I wanted to understand the impact to development times - without impact to performance - as we reduce the number of things a developer must do.

why is it hard:

1. While the more advanced programming languages come with lots of handy utlities and numerous frameworks , the developer has very little control over memory and type of object allocation.

2. These things make it challenging to get the same performance that we can get from a C based system.

why existing solutions do not work:

1. While its possible to write System programs in C - and C is the language of choice in this area. it is time consuming and prone to errors.

2. All C developers are familiar with the notorious runtime errors and core dumps that can take days to fix.

3. If the core language api can provide these general data structures and constructs , then developer can solely focus on the application development.

Architecture:

Scala:

1. Scala Tree is an interface that defines the contract that all implementations must comply with. It consists of a root - which is a type of TreeNode, children - that are List[BplusTree] and and optional Parent.

2. A tree can be - TypedTree or an EmptyTree

3. The TreeNode itself can be of Type LeafNode or a IndexNode

4. RichBplusTree - contains all operations that can be supported by a Tree - add, search

5. TreeFunctions - contain the utility methods such as print, toJson, sizeof

6. object BplusTree - has apply and unapply functions that encapsulate Tree Building and pattern matching utilities.

Add a List of elements

Step 1 - Find the Leaf that should contain this new element.

Step 2 – Add the element to this leaf Tree ( this add is the same add )

Step 3 – Remove the old leaf from the Tree. This returns a newTree without that LeafNode

Step 4 – Add back newLeaf to the Tree returned from operation 3. This step involves finding the parent that should contain this new LeafNode, adding the newLeaf, promote and dedupe , updating parents and finally ensure the tree is balanced. ( some parts of the code is shown below in the function addNodeAndRebalanceTree)

Step 5 – If a node is not the target – we copy it back in place in the new tree. This ensure that we always have all the Children of the Tree.

Range Search

Step 1 - find all qualifying parent and sibling nodes for the given criteria. This returns a list of BplusTree

Step 2 – Traverse the list of qualifying trees , walk to their leaves and collect the qualifying values.

Evaluation:

Set up:

1. The experimental platform consists x86\_64 machine with 4 cores running Intel(R) Xeon(R) CPU E5-1607 model, 32GB of memory.

2. The operating system is Ubuntu 12.04.2 LT.

3. The scala version used is 2.10.3 and the sbt version is 0.13.1.

4. The JVMs were configured to run with 12G of heap and Concurrent MarkSweep garbage collection strategy. For compiling C gcc standard c99 compiler was used.

5.The Fanout of the Tree was kept at 65 for all experiments.

6. Varying data sizes ( 100 to 100k integers) were loaded and queries were run for different selectivity

7. There is trend in the query graphs which show that scala querying is consistently better than C.

8. The size of the C tree is 10 time smaller than the scala tree and the tree build time for C stays pretty much constant, while for scala the increase is almost proportional to the increase in data size.

9.Studying the number of cache-misses, instructions for building the tree in scala – we can see that the number of instructions executed is really large for scala tree . This can be directly attributed to the way the scala tree is built – where each element added results in the whole tree being copied , so as the number of elements added increases so does the amount of copying and hence the garbage. This also triggers large GC cycles attributing to overall slow down of the whole build process.

Future Next steps:

1. In order to speed up the build times we need to reduce the amount of work down – which will reduce the number of instructions and bring down garbage collection cycles. One way to do this would be to “Bulk Load” with an immutable tree , or change the tree to be mutable just like C.
2. One thing that would be interesting to study would be the performance of mutable Scala Tree, built using a TreeBuffer ( very similar to the ListBuffer found in standard scala API).
3. It will also be interesting to compare the performance of a C , Scala and Java implementations. Since Java allows for unboxed types ( int, double).
4. Given the strong query performance of range queries in scala , another interesting area of study is the performance of scans in C and Scala