**Document for CS265 SDS Project: Exploring Typed Functional Language To Generate An Optimal B-Tree**

This document details the goals and design of developing a Polymorphic B-Tree in the context of ongoing research work on Self Designing Data System at DASlabs.

*Introduction:*

The goal of this project is to develop a polymorphic B-Tree and integrate it with an automatic iterative process, to generate an optimal B-Tree for a given set of conditions. The abstraction provided by such a Tree combined with the optimization algorithm makes it very adaptive design.

Applications usually have many different data types, and having a tree that can only be optimized for one data type would result in either developing a whole new tree and maintain separate code base for a each new data type that may be needed or using a sub-optimal tree.

*Background:*

*Self Designing Data Systems (SDS at Daslabs)* :

This project is a part of a much larger initiative of “Self Designing Data Systems” at Daslabs. The idea of Self Designing Data Systems is to develop system that can quickly generate an optimal system design for any given application. This not only reduces the time for coming up with optimal solutions for different application needs, but also helps to identify new design that can evolve from existing ones.

*Scala:*

This project will be implemented using [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) – a statically typed functional programming language which runs on the JVM. Using Scala’s rich type system we can use the compiler to provide guarantees of the invariant of a data type ( in this case B-Tree) at compile time (as oppose to runtime).

We will be using various functional patterns such as type classes, monads et al where applicable to provide the necessary abstractions ensuring that implementation (a) can be easily extended (b) is elegant and (c) exploits static type safety.

*Genetic Algorithm:*

We will use genetic algorithm to automate the processes of selecting the most optimal solution from amongst the available options.

*Design:*

This section presents the proposed design of the SDS B-Tree.

TreeOptimizer.scala

def generate[T](inputs) : BplusTree[T]

BplusTree.scala (abstract)

GATree.scala

RichBplusTree.scala

TreeFunction. scala

TypedTree[T]

EmptyTree

1. BplusTree.scala:

* Models the polymorphic BTree. It is implemented as a trait that is typed on T- an abstract type
* Designed using “type class” pattern – to generate different typed trees
* Tree is a recursive data structure that can be defined in terms of
  + EmptyTree – Models a tree that has no leaf or index nodes
  + TypedTree – Materializes an actual Typed Tree.

**trait** BplusTree[+T] {  
 **def** root: TreeNode[T]  
 **def** children: List[TreeNode[T]]  
}

//Empty Tree

**case object** EmptyTree **extends** BplusTree[Nothing] {  
 **val** *root*: TreeNode[Nothing] = *LeafNode*(*List*.*empty*, None)  
  
 **val** *children*: List[TreeNode[Nothing]] = *Nil*}  
  
*/\*\* Create a tree from base tree.  
 \*/***case class** TypedTree[T](elem: T, pos: Int, baseTree: BplusTree[T]) **extends** BplusTree[T] {  
  
 **lazy private val** *newTree* = baseTree **match** {  
  
 **case** a @ EmptyTree => **new** BplusTree[T] {  
 **val** *root* = *LeafNode*[T](*List*(*LeafValue*(elem, pos)), None, None)  
  
 **val** *children* = *Nil* }

**case** \_ => baseTree.add(elem, pos)  
 }

**val** *root*: TreeNode[T] = *newTree*.root  
 **val** *children*: List[TreeNode[T]] = *newTree*.children  
}

1. RichBplusTree.scala :

* Extends basic BplusTree and provides functions that are not present in the BplusTree
* Follows the Pimp My Library pattern – which extends a basic data types and adds new behavior for the data type.

*/\*\* Pimp my library pattern. Will be available implicitly on the Tree.  
 \** **@param tree** *\** **@tparam T** *\*/***class** RichBplusTree[T](tree: BplusTree[T]) {  
  
 **def** add[T](input: T, position: Int): BplusTree[T] = ???

**def** delete[T](input: T, position: Int): BplusTree[T] = ???

}

1. TreeFunctions:

* Modeled using the singleton pattern Tree functions – contains useful Tree function

*/\*\* Contains methods for working with the tree  
 \* Created by rsha on 2/27/16.  
 \*/***object** TreeFunctions {

**val** *FAN\_OUT* = 5

**def** children[T](t: BplusTree[T]) : List[TreeNode[T]]= *???*

**def** walk\_tree[T](t: BplusTree[T]) = *???  
 // side effecting printing operation* **def** pretty\_print[T](t: BplusTree[T]) {  
 t **match**{  
 **case** BplusTree.*empty* => ???

**case** \_ => *???* }

}

1. TreeOptimizer.scala :

* Parses the request input
* Dispatches request to GATree.scala , after transforming the request inputs to GAInput form.
* Receives the optimized tree from GATree for the inputs and prints it to the console

1. GATree.scala : (In progress )

* The class that models Genetic Algorithm used for coming up the optimal solution for given set of inputs
* Inputs that can be varied include
  + BplusTree type – can be any of String, Int, Double, Long, Dates, List[Int]…
  + LeafValue type – this models the container the holds the list of values at the leaf level. This is also typed – so can be anything that can be ordered – for example – Dates, Ints etc

**case class** LeafValue[T](t: T, p: Int) **extends** Ordered[LeafValue]

* LeafNode – LeafValues in the LeafNode can be modified to take any type that is an Iterable – In scala this would all collections like – Map, Set, List. For this we will change the definition of

LeafNode as below

**case class** LeafNode[T](leafValues: Iterable[LeafValue[T]],  
 parent: Option[IndexNode[T]] = None,  
 next: Option[LeafNode[T]] = None) **extends** TreeNode[T]

*Testing:*

* Unit test for all Tree functionality – such as add, delete
* Load testing of tree of type “ Int “ using data set from cs165
* Testing of Genetic Algorithm – and optimal solution

*Task breakdown and Timelines:* The below is based on the assumption of continuous feedback loop.

|  |  |  |  |
| --- | --- | --- | --- |
| Milestone | Description | Tentative Finish date / Total weeks | Status |
| Milestone1 | Initial setup –  Project structure, build and git repo | 28th Feb / 2 days | Complete |
| Milestone2 | Btree with add and delete with unit tests and load test ( CS165 data set) | 13th March/ 1.5 week | In progress |
| Milestone3 | Genetic Algorithm – basic implementation + unit test | 20th march/ 2 week | Not started |
| Milestone4 | Testing + bug fixes - GA with Tree | 16th April - 3 weeks | Not started |
| Milestone5 | Feedback + any new ideas | 24th April - | TBD |

*Git repo:*

git@code.seas.harvard.edu:cs265/cs265.git