In Memory Time Complexity of CASSI Trees

Deep Information Sciences: Thomas Hazel and Evan Lyle

Part I
Introduction

The simplest description of the efficiency of an algorithm is big-oh notation. This notation has become the standard method for determining an algorithm's limiting behavior (i.e. cost). The "cost" of an algorithm is often described in one of two ways, memory or disk cost. This paper will focus on the memory cost of the CASSI tree, but will touch on disk cost as well. Big-oh (notated O) is akin to an upper bound on the algorithm as n increases where n is the input to the algorithm. Big-oh is formally defined as follows:

$$f(n) = O(g(n)) \text{ if } \exists \text{ constants } C, N \text{ such that } \forall n \text{ with } n > N, \mid f(n) \mid \leq C \cdot \mid g(n) \mid$$

Big-oh notation has become the de facto method for describing database operational cost where the B-tree data structure is typically the algorithm of choice. This type of algorithm operates in $O(log(n))$ time and is considered to be highly efficient. However, big-oh notation leaves out constants, which can greatly affect the overall run time of an algorithm. For instance, the CASSI (Continuous Adaptive Sequential, Summarization of Information) tree also has $O(log(n))$; yet, because O notation leaves off constants, CASSI trees generally run faster than B-trees in implementation.

Part II
B-Trees

The B-tree is one of the most commonly used structures for ordered data storage. This is because B-trees are predictable and at the time of their inception, comparatively fast. B-trees use a sequential search to find objects within the tree. The B-Tree is $O(log(n))$ for C.R.U.D. (create, read, update, and delete) actions. However, the run time with constants included is $(\frac{t}{log(t)}) \cdot log(n)$ where t is the constant degree of the B-tree. The degree of a B-tree is the number of elements stored in each branch of the tree and is not a function of n. The proof

of B-tree run time is as follows:

1. The total CPU time of a search for a B-tree is $O(t \cdot h)$ where $t > 2$ is the constant degree of the B-tree and h is the height.
2. Each level of the tree takes time t to search because there are t items on each branch and thus t comparisons.
3. A standard B-tree has height $O(log_t(n))$ and therefore takes time $O(t \cdot log_t(n))$
4. $t \cdot log_t(n) = \frac{t}{log(t)} \cdot log(n)$

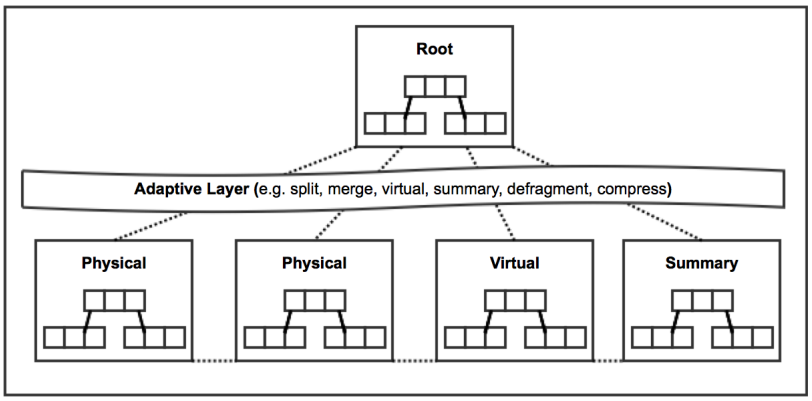
The run time is calculated by multiplying the height (how many levels the B-tree is made up of) of the tree by the time each search will take. This makes sense since the algorithm only makes one search per level. This proof shows that even though B-trees are $O(log(n))$, there is room for improvement in time complexity.

Part III
CASSI Trees

1 Introduction

The CASSI tree can manifest as in-memory and on-disk algorithms and data structures. The following describes CASSI's in-memory time complexity. Some aspects of CASSI will be deffered to "On disk Time Complexity of CASSI Trees" paper.

The CASSI tree is organized in a tree-like structure comprised of a "Root" and numerous branches. The branches of the tree fall into three categories: Physical, Virtual, and Summary. CASSI trees have the ability to collapse internal structure by way of virtualizing and summarizing. This collapsing is facilitated by separating the "Root" branches from subsequent branches called "Segments", all of which allow for intelligent organization of information (i.e. adaptive). These components can be thought of as branches of a larger tree stemming from The Root. However, these components are not necessarily linearly connected like a B-tree would be. The diagram below shows the basic structure of a logical CASSI tree:



In addition to the ability to intelligently collapse and summarize information, CASSI trees also offer innovation in the use of binary search and "add". Accompanying traditional C.R.U.D. operations, CASSI also includes a highly optimized operation called "add". This operation is like an insert but for data that is known to be sequential. In other words, the "add" operation does not have to make any compares while constructing the tree because the data is sequential and thus has a cost of $O(1)$.

2 Binary Search

CASSI Trees use a binary search to find items within the tree. The binary search property of a CASSI tree lowers the run time compared to a B-tree because it makes the run time independent of the degree of the tree even though CASSI trees still have $O(log(n))$ time complexity. In a B-tree, each level of the tree is searched sequentially until the object of the search is found. In a CASSI tree, each level is searched logarithmically. That is to say, instead of searching the $1st$ item, $2nd$ item, etc, CASSI trees jump to the middle item which splits the set in half until it finds the item. Binary search algorithms have a cost of $O(log(n))$. This is an improvement over sequential search and eliminates a constant that affects overall run time. The proof of this is as follows:

1. The CPU time of a CASSI tree search is $O(log(t) \cdot h)$ where t is the degree (number of items in each branch) of the tree (the $log(t)$ represents the time to search each level of the tree because of the binary search property of CASSI trees) and h is the height of the tree
2. $h = O(log_t(n))$
3. The CPU time of a CASSI tree search can be re-written as $O(log(t)(log_t(n)))$
4. This simplifies to $O(log(t) \cdot \frac{log(n)}{log(t)})$ and finally $O(log(n))$

This proof shows that, by using a binary search, CASSI trees can improve on B-trees' run time by making the run time no longer affected by t. The binary search also has a best case which is covered in the charts section. CASSI trees also offers a number of formations (i.e. virtual, summary) in which they can improve performance over B-trees.

3 Virtual Segments

The Virtual Segment is a branch classification of the CASSI tree. These Segments are a case where the CASSI tree can improve overall performance. The Virtual Segment is a logical representation (i.e. not fully instantiated) of a Segment where some C.R.U.D. operations can still be performed without constructing the entirety of the Segment from disk. It follows that the best case time complexity of a Virtual Segment is $O(1)$. A proof is included below:

1. The best case complexity for a virtual Segment is when the physical size (versus logical size) of the Segment has a height of zero (e.g. only one level).

2. Physical size is the actual elements contained by the Segment (i.e. what's in memory) and the logical size is the virtual number of elements that if fully constituted from disk would be the entire Segment.

3. Any portion of elements less than the fully constituted size makes a Segment "Virtual". This gives Virtual Segments a best case of $O(1)$. because the size of each level is fixed and not dependent on n.

4 Physical Segments

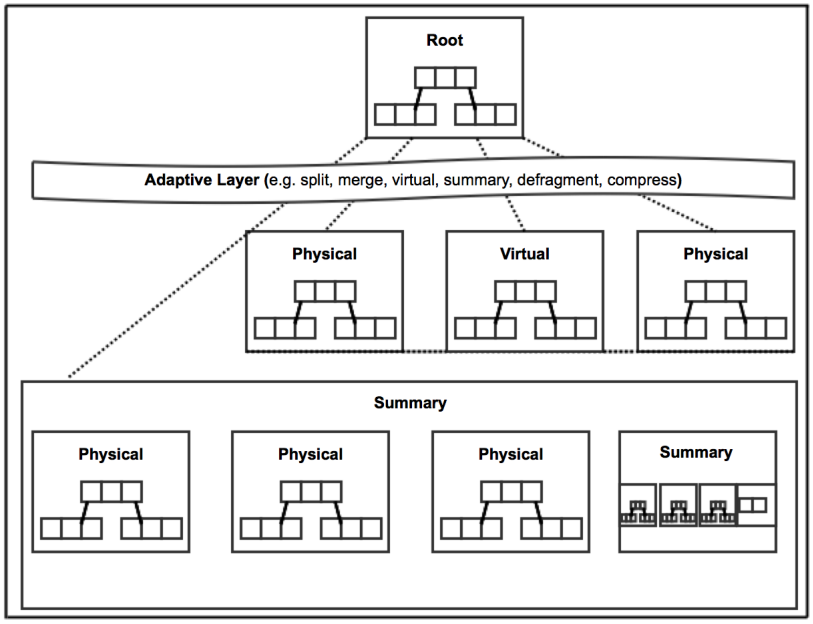
The Physical Segment is another branch classification of the CASSI tree. Physical Segments are fully instantiated CASSI tree with all items contained, unlike its Virtual and Summary counterparts. This is the most basic type of Segment and is most analogous to B-tree paging in a traditional database.

5 Summary Segments

A Summary Segment is a logical aggregation of subsequent Segments, physical or summarized, that works to manifest a fully substantiated CASSI tree (i.e. Root and branches). The purpose of a Summary Segment is to collapse large regions of the key-space. It can be seen as a logical representation (i.e. not fully instantiated) of a multiple contiguous Segments where some read operations can be performed without constructing the entirety of the Segments from disk.

In addition to optimizing read operations, Summary Segments provide useful statistics (metadata) as well as locality of data on disk. Such statistics can be used in conjunction with other systems such as query optimizers. Many of the other optimizations offered by Summary Segments are manifested on disk such as cold start discovery and recovery procedures. Summary Segments improve read performance even though they have $O(log(n))$ cost by minimizing the inputs to the tree.

Below is a diagram of a CASSI tree containing a Summary Segment:



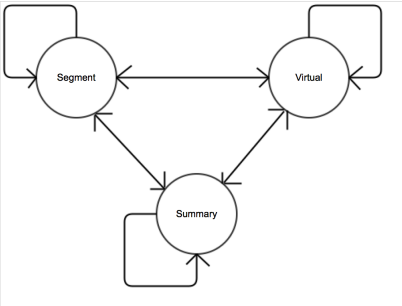
6 Adaptive Layer

The adaptive layer lives between the root of the tree and its branches (i.e. Segments). This layer uses machine learning concepts to optimize the algorithm by observing and predicting the application workload. This maximizes hardware resources to improve overall run time. Although this layer is similar to some adaptive sorting algorithms, the adaptive layer does not need to make n comparisons because the data is assumed to already be in perfect order. Since the adaptive layer operates independently of n, it does not affect the O notation for basic operations of the tree but it does offer significant improvements in overall run time. The adaptive layer orchestrates the subsequent layers of the tree to other limiting factors as opposed to the ordering of the data.

The adaptive layer can be thought of as a sort of genetic algorithm for all of the Segments of the tree. The adaptive layer determines all of the characteristics of the tree. The overall look and function of the CASSI tree is dependent on the decisions made by the adaptive layer. For example, the adaptive layer sets the size, type (i.e. Virtual, Summary), and order for each of the Segments. Also, adaptive makes decisions about when and if to change the type of Segments (e.g. changing a Physical Segment to Virtual Segment). The purpose of these decisions is to minimize resources used for operations.

The adaptive layer makes predictions about which parts of the tree are likely to be "hot" in the future and adjusts the tree accordingly. Adaptive can be described using a Markov Decision Process. This statistical technique is used to model processes that have different states. For example, data that is currently being operated on ("hot") can either continue being operated on, or stop

("cold"). The frequency with which a chunk of data is operated on is described as the "weight" of a Segment where weight is the rate of change. By measuring the probabilities of each of these events, adaptive takes the data that is most likely to become hot or remain hot and ranks it in order according to weight. The algorithm then uses these predictions to greatly increase performance by allocating hardware resources accordingly. This same process is also used to determine which state each Segment should be in. A diagram of this property is shown below. Each Segment can either remain the same or change to any other state.



Part IV
Summary

Although the B-tree was revolutionary at its inception, it can be improved upon. The Even though CASSI trees have the same $O(log(n))$ time complexity, CASSI trees offer significant improvements in run time. The improvement in run time can be attributed to numerous innovations in the structure of the tree such as the implementation on binary search. The Virtual and Summary Segments maximize hardware resources and the adaptive layer makes predictions and decisions based on a number of different statistics generated in a Markov model to optimize overall performance.

Part V
Charts

Segment Big-Oh

	Best	Average	Worst
Add	O(1)	O(1)	O(1)
Insert	O(log(n)/log(t))	O(log(n))	O(log(n))
Update	O(log(n)/log(t))	O(log(n))	O(log(n))
Delete	O(log(n)/log(t))	O(log(n))	O(log(n))
Read	O(log(n)/log(t))	O(log(n))	O(log(n))

Virtual Segment Big-Oh

	Best	Average	Worst
Insert	O(1)	O(log(n))	O(log(n))
Update	O(1)	O(log(n))	O(log(n))
Delete	O(1)	O(log(n))	O(log(n))
Read	O(log(n))	O(log(n))	O(log(n))

Summary Segment Big-Oh

	Best	Average	Worst
Insert	n/a	n/a	n/a
Update	n/a	n/a	n/a
Delete	n/a	n/a	n/a
Read	O(log(n))	O(log(n))	O(log(n))

Part VI
References

Cormen, Thomas H., Charles Eric. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Third ed. Cambridge, MA: MIT, 1990. Print.

Neubauer, Peter. "B-Trees: Balanced Tree Data Structures." B-Trees. N.p., 1999. Web. 23 June 2015.

Larsen, Kim S., and Rolf Fagerberg. "Efficient Rebalancing of B-Trees With Relaxed Balance." *International Journal of Foundations of Computer Science* (n.d.): n. pag. Web. 23 June 2015. < Larsen, Kim S., and Rolf Fagerberg. International Journal of Foundations of Computer Science (n.d.): n. pag. Web. 23 June 2015. >

Neylon, Tyler. "Big-Oh For Algorithms." *Medium*. N.p., 22 Jan. 2014. Web. 23 June 2015.

Murphy, Kevin P. *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT, 2012. Print.

Feitelson, Dror G. *Workload Modeling for Computer Systems Performance Evaluation*. N.p.: n.p., n.d. Print.

Wan, Lipeng, Zheng Lu, Qing Cao, Feiyi Wang, Sarp Oral, and Bradley Settlemyer. "SSD-optimized Workload Placement with Adaptive Learning and Classification in HPC Environments." *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)* (2014): n. pag. Web.