B+ TREES

Ajay, Manjunath
Vaibhav

Thursday 18 March 2010 06:44:22 PM IST

1 DESCRIPTION

B+ Trees
In computer science, a B+ tree (BplusTree) is a type of tree which represents sorted data in a way that allows for efficient insertion, retrieval and removal of records, each of which is identified by a key. It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each index segment (usually called a "block" or "node"). In a B+ tree, in contrast to a B-tree, all records are stored at the leaf level of the tree; only keys are stored in interior nodes.

The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context in particular, filesystems. This is primarily because unlike binary search trees, B+ trees have very high fanout (typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

Relational database management systems such as IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASI, PostgreSQL, Firebird and MySQL support this type of tree for table indices. Key-value database management systems such as Tokyo Cabinet and Tokyo Tyrant support this type of tree for data access.

1.1 TIME AND SPACE COMPLEXITY

For a b-order B+ tree with h levels of index:
The maximum number of records stored is nmax = bh
The minimum number of keys is nk = (b / 2)h 1
The space required to store the tree is O(n)
Inserting a record requires O(logbn) operations in the worst case
Finding a record requires O(logbn) operations in the worst case

2 ALGORITHM

Algorithm 1 ALGORITHM FOR INSERTION

if $(ROOT \leftarrow null)$ **then**
  $ROOT \leftarrow [key]$
end if
$INCREMENTCOUNTandRETURNROOT$
$LEAF \leftarrow FINDLEAF(ROOT, key)$
if $(LEAFHASSPACE)$ **then**
  $INSERTLEAF(LEAF, key)andRETURNROOT$
  $RETURNSPLITLEAF(ROOT, LEAF, key)$
end if

Algorithm 2 ALGORTIHM TO FIND LEAF

$NYPEfindleaf(ROOT, key)$
if $(keyexistsinNODE)$ **then**
  $ERRORMESSAGE \leftarrow Noduplicatesallowedandexit(0)$
end if
if $(ROOT \leftarrow leaf)$ **then**
  $RETURNROOT$
end if
$FINDPOSITIONOFLINK$
$FINDLEAF(ROOT \leftarrow LINK, key)$
$NTYPEINSERTLEAF(LEAF, key)$
$FINDPOSITIONWHERETHEKEYHASTOBEPLACED$
$SHIFTELEMENTSAHEADTOMAKEROOMFORKEY$
for $(i \leftarrow LEAF \rightarrow COUNTtoPOSITION)$ **do**
  $LEAF \rightarrow [i] = LEAF \rightarrow key[i-1]$
end for
$LEAF \rightarrow KEY[POSITION] = key$
$INCREMENTCOUNT$
$RETURNCOUNT$

Algorithm 3 AGORITHM TO SPLIT LEAF

$NTYPESPLITLEAF(ROOT, LEAF, key$
$COPYTHECONTENTSOFLEAFTOTEMP$
for $(i \leftarrow toLEAF \rightarrow COUNT)$ **do**
  $TEMP[i] = LEAF \rightarrow key[i]$
end for
$FINDPOSITIONWHEREKEYSHOULDBEPLACES$
$SHIFTELEMENTSTOMAKEROOMFORKEY$
while $(positionislessthanorder - 1)$ **do**
  $temp[k] = temp[k-1]$
  $DECREMENTk$
end while
$TEMP[POSITION] \leftarrow= key$
$DETERMINESPLITLENGTH[(n/2)]$
$PUTVALUESINTOLEAF$
for $(i = 0tolength)$ **do**
  $LEAF \rightarrow key[i] = TEMP[i]$
end for
$PUTVALUESINTONEWLEAF$
for $(i = lengthtoorder)$ **do**
  $NEWLEAF \rightarrow KEY[i] = TEMP[i]$
  $INCREMENTj$
end for
$NEWLEAF \rightarrow PARENT = LEAF \rightarrow PARENT$
$NEWLEAF \rightarrow SIBLINGPTR = LEAF \rightarrow SIBLINGPTR$
$VAR = NEWLEAF \rightarrow key[0]$
$RETURNINSERTPARENT(ROOT, LEAF, NEWLEAF, VAR)$

Algorithm 4 ALGORITHM TO INSERT PARENT

$NTYPEINSERTPARENT(ROOT, LEAF1, LEAF2, VAR)$
$PARENT = LEAF1 \rightarrow PARENT$
if $(PARENT = NULL)$ **then**
  $PARENT \rightarrow key[0] = VAR$
  $PARENT \rightarrow ptr[0] = LEAF1$
  $PARENT \rightarrow ptr[1] = LEAF2$
  $LEAF1 \rightarrow = LEAF2 \rightarrow PARENT$
  $RETURNPARENT$
end if
$DETERMINEPOSITIONOFLINKFROMPARENTTOLEAF1$
$SHIFTPOINTERSANDKEYSAHEADTOMAKESPACE$
for $(i = PARENT \rightarrow counttoPOSITION)$ **do**
  $PARENT \rightarrow key[i] \rightarrow PARENT \rightarrow key[i-1]$
  $PARENT \rightarrow ptr[i+1] = PARENT \rightarrow ptrr[i]$
end for
$PARENT \rightarrow ptr[POSITION + 1] = PARENT \rightarrow ptr[POSITION]$
$PARENT \rightarrow key[POSITION] = VAR$
$PARENT \rightarrow ptr[POSITION + 1] = LEAF2$
$RETURNROOT$
$RETURNSPLITPARENT(ROOT, PARENT, LEAF2, POSITION, VAR)$

Algorithm 5 ALGORITHM FOR SPLIT PARENT

$NTYPESPLITPARENT(ROOT, PARENT, LEAF2, POSITION, VAR)$
$COPYPOINTERSOFKEYSOFPARENTTOTEMP$
for $(i \leftarrow 0toorder - 1)$ **do**
  $TEMP(POINTER[i]) = PARENT \rightarrow ptr[i]$
  $TEMP(POINTER[i]) = PARENT \rightarrow key[i]$
end for
$TEMP(POINTER[order - 1]) = PARENT \rightarrow ptr[order - 1]$
$SHIFTPOINTERS, KEYSAHEADTOMAKESPACE$
for $(i \leftarrow order - 1toPOSITION)$ **do**
  $TEMP(POINTERS[i + 1] = TEMP(POINTERS[i])$
  $TEMP(KEY[i]) = TEMP(KEY[i - 1])$
  $TEMP(POINTER[POSITION + 1]) = LEAF2$
end for
$TEMP(KEY[POSITION]) = VAR$
$ERASEPARENT$
$COPYFIRSTHALFOFPOINTER, KEYTOPARENT$
for $(i \leftarrow toPlength)$ **do**
  $PARENT \rightarrow ptr[i] = TEMPPOINTER[i]$
end for
$CPOYNEXTHALFOFTHEPOINTER, KEYTONEWNODE$
for $(i = Plengthtoorder - 1)$ **do**
  $NEWNODE \rightarrow ptr[j] = TEMPPOINTER[i]$
  $INTERMEDIATEVALUETOPASS$
  $Pvar = TEMPKEY[Klength]$
  $ConnectchildnodeofNEWNODEtoNEWNODE$
end for
for $(i = 0toNEWNODE \rightarrow COUNT)$ **do**
  $NEWNODE \rightarrow ptr[i] \rightarrow= NEWNODE$
end for
$RETURNINSERTPARENT(ROOT, PARENT, NEWNODE, pvar)$

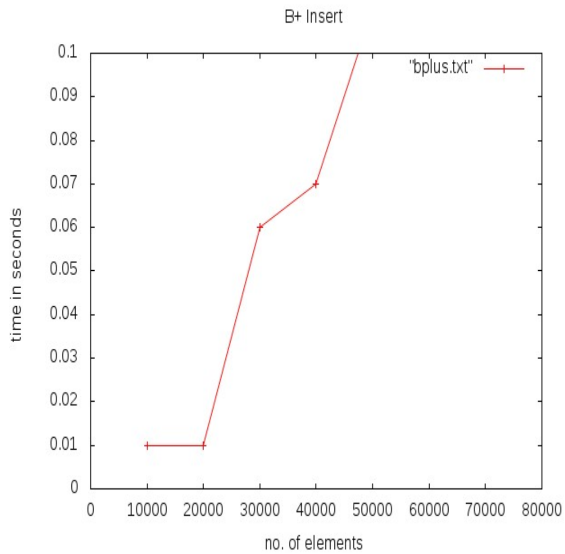

Figure 1: Graph Plot Of B+ Trees

3 PROFILING

Table 1: B+ trees

no of elements	time taken in secs
10000	0.020000
20000	0.020000
30000	0.040000
40000	0.070000
50000	0.100000

4 CONCLUSION

1. We successfully implemented B+ tree insertion which is used in various disk systems.
2. Our code has three versions viz. 1.User-interface 2.Dotty-interface 3.Graph-plot using random function code.
3. Looking at the above graph we can conclude that it takes very less time to insert/ access the data in the tree.
4. To insert the first 10,000 -20,000 elements the graph value was a constant. After 20,000 + elements the growth was linear.
5. This project helped us to re-inforce our programming skills.
6. It also helped us to know the power of Linux(Ubuntu).