AN ANALYTIC MODEL FOR PERFORMANCE EVALUATION OF CONCURRENCY CONTROL ALGORITHMS

J. K. Chen
Department of Information Management
Chaoyang University of Technology, Wufeng, Taichung County, Taiwan, R.O.C.
Email: jkchen@mail.cyut.edu.tw

ABSTRACT

An analytic model is proposed to explore performance of a concurrency control algorithm (CCA) of access method. Three dominant factors can indicate whether a CCA is better (or worse) than another in a multi-user environment. These factors reflect the methods for search and concurrency control of a CCA. To make a criterion for evaluating a CCA in term of these three factors, an analytic model is derived to formularize each value of each factor for the worst case. By the model, we can evaluate the performance of each CCA and identify which CCA has the best performance. To illustrate the applicability of the model, four CCAs for R-trees are used as examples to explain how to compute the values of these three factors of a CCA through the analytic model.

KEYWORDS: Performance evaluation, Analytic model, Access method, Concurrency control

1. INTRODUCTION

The performance of a CCA is usually evaluated based on throughput, response time, or both [1], [2], [6], [7], [21], [23], [24]. However, these two metric criteria can be obtained from either a practical or a simulation database system and that will take much time. Another quick method is using an analytic model [12], [15], [24] and this paper is presented for this purpose. Many factors in fact influence the response time or throughput of a CCA. Some are related to the access method, the underlying data structure, and the protocols for locking/unlocking a data object. Among them, three main factors have dominant influence in the performance of a CCA, namely, the number of accessed nodes, the number of locked nodes, and the locked range [10]. Basically, these factors can be measured through analyzing the properties of a CCA. Therefore we can compute the values of the three factors for each CCA and identify a CCA that has the best performance.

To make a criterion in term of these three factors, we propose an analytic model that can formularize each value of each factor for a special access method that uses a certain data structure and/or locking protocol. Then we total the values of the three factors as a standard value. This standard value is used for comparison with other summation value of the three factors of an evaluated CCA. If we want to select a CCA that has the best performance from some evaluated CCAs, we first compute the summation value of the three factors for each CCA. Then each difference between each summation value for each CCA and the standard value created by the analytic model is computed. Finally the CCA with the smallest difference is identified. To illustrate the applicability of the analytic model, two couples of proposed R-tree [11] CCAs are selected as examples since the R-tree family [3], [11], [20] are popular in many applications [5], [19], [22]. Figure 1 shows an R-tree for illustration. The first couple of CCA examples are NK (proposed by Ng and Kameda [16]) and CHC (proposed by Chen, Huang, and Chin [9]). Their locking protocol is based on the technique of lock-coupling [4]. The second couple of CCA examples are KB (proposed by Kornacker and Banks [13]) and CC (proposed by Chen and Chin [8]) that use the right-linking technique [14] as their locking protocol[1]. As a result, CHC (CC) has better performance than NK (KB) because the difference between the values of the three factors for CHC (CC) and the standard value is smaller than that for NK (KB).

2. THREE FACTORS

The semantic definitions of the three factors are described as follows. First, the number of accessed

[1] The CCA [17] cannot be used since its insertion algorithm is unavailable.

nodes (AN) indicates how many nodes are accessed by an evaluated operation. An operation may take a long time to access nodes if the value of AN is large. Second, the number of locked nodes (LN) describes the total number of nodes that have ever been locked by an evaluated operation. The larger the value of LN, the longer the waiting and locking time. Third, the locked range (LR) represents the largest number of nodes that were locked at the same time by an evaluated operation. This factor reflects the phenomenon of resource holding made by a concurrent operation, and it influences the liquidity of other concurrent operations. The larger the value of LR, the lower the degree of concurrency. These three factors are directly relevant to an access method and the concurrent control protocol used by the access method. In general, the smaller the values of these factors of a CCA, the better the performance of the CCA.

As the definitions of these factors stated, both AN, LN are related to the response time. LR has an influence on both the response time and throughput. Therefore, these three factors can be used as basic indicators to verify the performance of a CCA. An analytic model is then derived for evaluating the performance of a CCA. For illustration, Section 4.1 shows how to apply the the analytic model to identify the desired CCA using the lock-coupling locking protocol. Section 4.2 presents the same procedure to identify the desired CCA using the right-linking locking protocol.

3. ANALYSIS MODEL

The values of the three factors are analyzed and the corresponding formula for each factor is derived for *the worst case*. Different locking protocols produce different values of the factors in a CCA. To be a representative, two popular locking protocols: lock-coupling [4] and right-linking [14] are selected. The fundamental difference between lock-coupling and right-linking is as follows. The technique of lock-coupling is to release the lock on the parent node as soon as the locking request of its child node was granted by the lock manager. In this way, the child node is guaranteed to be intact before visiting this child. Contrarily, right-linking will first release the lock on the parent node, and then lock the target child node. This technique uses the right-link pointer to find the correct node if the node-missing problem occurs. Without loss of generality, there are three assumptions in the R-tree as follows. First, the maximum fanout of a node is $\boldsymbol{M}$ which indicates that a node may have at most M children and at least $\boldsymbol{m}$ children, where $m \geq \lceil M/2 \rceil$. Second, the height of the R-tree is $\boldsymbol{h}$. Third, the R-tree is leveled in an ascending order from root to leaves; therefore, the levels are numbered 0, 1, 2, ..., h-1. The analysis for each methodical factor is in the following.

3.1 AN Factor

In a search operation with either lock-coupling or right-linking, all the nodes in the R-tree will be visited when the search range is large enough to cover all data objects in the tree for the worst case. The value of AN for such a search operation is the total number of nodes in the R-tree. The total number of nodes can be estimated by computing the average of the maximum and the minimum of nodes. In the case of maximum, each non-leaf node has M children. Since the height of the R-tree is h, the total number, say, $N\text{-}max$, of nodes in the R-tree can be derived approximately as follows: $N\text{-}max = 1 + M + M^2 + M^3 + ... + M^{h-1} = (M^h - 1) / (M - 1)$. In the case of minimum, each non-leaf node has m children and the root has only two children. The total number, say, $N\text{-}min$, of nodes in the R-tree can be derived approximately as follows: $N\text{-}min = 1 + 2 + 2m + 2m^2 + ... + 2m^{h-2} = 1 + 2(m^{h-1} - 1) / (m - 1)$. Therefore, the formula for computing the average value of AN is $(N\text{-}max+N\text{-}min) / 2 = [(M^h\text{-}1)/(M\text{-}1) + 1 + 2(m^{h-1}\text{-}1)/(m\text{-}1)] / 2 = 1/2 + (M^h\text{-}1)/[2(M\text{-}1)] + (m^{h-1}\text{-}1)/(m\text{-}1)$. In an update (insert/delete) operation with either lock-coupling or right-linking, we only descends along a path to a target leaf node. The value of AN in an update operation can be described as follows. To traverse the path, we need to access h nodes. The ancestors of the target leaf node must be accessed again if the leaf node is split and the split propagates upward to the root. To handle the split propagation, we must re-access h-1 nodes (the ancestors of the leaf node in the path)[2]. Besides, there are h new nodes to be created as twins of the nodes in the path. Finally, a new root node must be created.

[2]For simplicity, we do not consider the extra accessed nodes due to the node-missing problem.

Thus, the formula for the value of AN is $h + (h - 1) + h + 1 = 3h$.

3.2 LN Factor

In a search operation with lock-coupling, each node must be locked before being visited. All the nodes in the R-tree must be locked at least once if the search range is large enough to cover all data objects. Therefore, the formula for the value of LN is the same as that of AN in the search operation, namely, $1/2 + (M^h-1) / 2(M-1) + (m^{h-1}-1) / (m-1)$. Contrarily, a search operation with right-linking has no need to lock any node before visiting that node because the right-link pointer can solve the node-missing problem. Hence, the formula for the value of LN is 0.

In an update operation with lock-coupling, each node should be locked before being visited when descending along a path to a leaf node. The value of LN is similar to that of AN in an update operation. However, we have no need to lock the h new twins of the nodes in the path we pass because the twins are isolated when they are created. Thus, we need to lock h nodes, re-lock h-1 nodes (ancestors of the target leaf node), and lock the new root. The formula for the value of LN is $h+(h-1)+1 = 2h$. As for an update operation with right-linking, the answer is different to $2h$. Except for the leaf node, we can omit locking h-1 ancestors of the target leaf node when descending alone a path to the leaf node. This is because we can correctly reach the target leaf node by right-link pointers, if necessary. However, we still need to lock the ancestors of the leaf node if the leaf node is split and the split propagates upward to the root. Therefore, the formula for the value of LN is $1+(h-1)+1 = h+1$.

3.3 LR Factor

In a search operation with lock-coupling, LR has different values according to the search method used in the CCA. If depth-first search is used, we may need to lock all the nodes along a path. The value of LR may be h. If breadth-first search is used, the situation of maximal locked nodes appears when a certain node is being visited and all its children are selected, locked to be visited later. The value of LR may be M+1. In general, M is larger than h because h is usually not more than 5 as stated in [18]. Thus, the formula for the value of LR is defined to be M+1. Obviously, the formula for the value of LR in a search operation with right-linking is 0 since the search operation does not lock any node when searching the tree.

In an update operation with lock-coupling, the formula for the value of LR is 2 and it occurs when the child node is locked successfully while the parent node is not released yet. As for an update operation with right-linking, the formula for the value of LR is 1 because a node is locked only if it will be modified immediately and is released right away after the modification. To summarize the analysis, the formulas of the methodical factors are shown in Tables 1 and 2.

TABLE 1
The Formulas of the Methodical Factors for CCA with Lock-Coupling in the Worst Case

	AN	LN	LR
Search Operation	$1/2 + (M^h-1)/[2(M-1)] + (m^{h-1}-1)/(m-1)$	$1/2 + (M^h-1)/[2(M-1)] + (m^{h-1}-1)/(m-1)$	$M + 1$
Update Operation	$3h$	$2h$	2

TABLE 2
The Formulas of the Methodical Factors for CCA with Right-Linking in the Worst Case

	AN	LN	LR
Search Operation	$1/2 + (M^h-1)/[2(M-1)] + (m^{h-1}-1)/(m-1)$	0	0
Update Operation	$3h$	$h + 1$	1

4. ILLUSTRATIVE CCAS

Two couples of CCAs are selected as illustrative examples to describe how to identify a well-performance CCA through the analytic model. The criterion for selection was based on the popularity of techniques of concurrency control protocols. The first couple of CCA examples, based on the popular *lock-coupling* technique, are NK (proposed by Ng and Kameda [16]) and CHC (proposed by Chen, Huang, and Chin [9]). The second couple of CCA examples, adopting the *right-linking* technique, are KB (proposed by Kornacker and Banks [13]) and

CC (proposed by Chen and Chin [8]). Subsequently, the search and update (insert/delete) operations based on the corresponding algorithms in the four CCAs are briefly described. The values of the three factors for each CCA are estimated for the worst case.

4.1 Lock-coupling CCA

For a search operation, NK or CHC usually lock-couples along multi-paths, starting from the root to the desired leaf nodes. They use the same search method, the same data structure of tree, and the same locking protocol. Therefore the values of the three factors of both NK and CHC are the same as the standard ones for the worst case, namely, $1/2 + (M^h-1)/2(M-1) + (m^{h-1}-1)/(m-1)$, $1/2 + (M^h-1)/2(M-1) + (m^{h-1}-1)/(m-1)$, and $M+1$.

For an update operation, NK or CHC first lock-couples along a path downward to find an appropriate leaf node and builds a scope[3] for reconstruction if necessary. After inserting (deleting) the object into (from) the desired leaf node, the update operation adjusts the MBRs (maximum bounding rectangles) of the leaf node and its ancestors in ascending order, reconstructing the tree if overflows or underflows occur. Likewise, NK and CHC use the same search method, the same data structure, and the same locking protocol to update a data object, but the process of overflow (underflow) is different. The values of the three factors of NK are larger than those values of CHC when overflows (underflows) occur. The reason is because NK accesses and locks concurrently several nodes for overflow (underflow) processing. When an overflow (underflow) occurs, NK always overlays three adjacent levels of tree and exclusively locks relevant nodes on these three levels. The involved nodes are (1) the overflow (underflow) node itself, (2) its parent node, and (3) all its child nodes. The values of AN, LN, and LR of NK are increased as follows. There are 1, $(M+1)(h-2)$, M extra nodes must be accessed for the overflow (underflow) processing of the leaf node, non-leaf nodes, the root, respectively. The value of AN is increased to $3h + 1 + (M+1)(h-2) + M = 3h + (M+1)(h-1)$. The number of extra nodes to be locked for the overflow (underflow) processing is the same as that of extra accessed nodes. Thus the value of LN is also increased to $3h+(M+1)(h-1)$. As to the value of LR, it is $M+2$ because the overflow (underflow) node itself, the parent node, and all the child nodes are all locked concurrently at a time.

As for CHC, it handles the overflow (underflow) by exclusively locking only the overflow (underflow) node and one of its child nodes. The number of locked adjacent levels can be reduced from three to two and only two nodes are involved at each overflow (underflow) processing. Only h-1 extra nodes must be accessed and locked for the overflow (underflow) processing of the nodes in the path. The value of AN is increased to $3h + h - 1$. The value of LN is also increased to $3h+h-1$. As to the value of LR, it is 2, equaling the standard value, because only two nodes are involved at each overflow (underflow) processing. Figure 2 gives an example about this situation. Assume an object N is inserted into the leaf node *f*. Node *f* is split into two nodes, *f* and *j*, due to an overflow. The overflow propagates upward to node *b*. When node *b* is being split, CHC only locks nodes *b* and *f*, which overlay two adjacent levels of tree, while NK locks all related nodes *a*, *b*, *d*, *e*, and *f*, which overlay three adjacent levels of tree. Now we compute the difference between the standard summation value and the summation value of the three factors for CHC and NK. The difference for NK is $[0+0+0] + [(M+1)(h-1)+(M+1)(h-1)+(M+2)] = 2(M+1)(h-1) + M + 2$. The difference for CHC is $[0+0+0] + [(h-1)+(h-1)+0] = 2(h-1)$. As a result, CHC has better performance than NK does since $2(M+1)(h-1) + M + 2$ is larger than $2(h-1)$.

4.2 Right-linking CCA

A right-link CCA applies a right-link pointer to each node of an R-tree. A right-link pointer points to its right sibling as the B^{link}-tree does [14]. However, the property of *high key* in B^{link}-trees does not exist in R-trees because keys in a B-tree are ordered linearly, while MBRs in R-trees have no such a property. To detect that the right margin is reached when moving from one node to its right siblings by right-link pointers, KB uses a time-stamp-like data item: LSN (logical sequence number) [13] while CC uses a pointer called delimiter [8].

For a search operation, KB descends down the tree along multiple paths from the root to the desired leaf

[3] In a path, a chain of nodes modified by an update operation is called the scope of the update operation [21].

nodes and uses the right-link pointer to solve the node-missing problem. The value of AN for KB is equal to the standard value. However, the values of LN and LR for KB are increased because KB locks each visited node and more than one node may be locked at a time when the node-missing problem occurs. Such a locking method does not sufficiently utilize the property of the right-linking technique to achieve the goal of a lock-freedom method as described in [14]. The value of LN is increased to 0 + $[1/2+(M^h-1)/2(M-1)+(m^{h-1}-1)/(m-1)]$. The value of LR is increased to 0+1. Contrarily, CC does the same searching operation as the B$^{\text{link}}$-tree does without locking any node [14]. In this way, the values of AN, LN and LR for CC are the same as the standard values of the three factors. For example, Figure 3 shows the location of the search window of a search operation. The target objects are G and K. KB's search operation accesses and locks at least 5 nodes, *a*, *b*, *c*, *f*, and *h*, if other nodes accessed and locked during right-link navigation are not counted for ease of illustration. Following the search operation based on the search algorithm in CC, these 5 nodes are only accessed without any requirement of locking them.

For an update operation, KB performs the same way that a lock-coupling CCA does without building a scope. KB solves the node-missing problem by using the right-link pointer like its search operation does. We observe that the value of AN for KB is the same as the standard value of AN, but the values of LN and LR for KB are increased due to the following two reasons. First, KB locks each visited node when descending along a path as its search operation does. This method also violates the goal of the lock-freedom property supported by the right-linking technique [14]. There are h-1 extra nodes to be locked. The value of LN is increased to $h + 1 + (h-1)$. The value of LR is no changed. Second, KB may lock three nodes simultaneously when ascending a tree path if overflow and the node-missing problem occur. Figure 4 shows an example of reconstructing the tree after an object N is inserted into the leaf node *f*. When the overflow of node *f* propagates to node *b*, KB will lock node *f* and the twin nodes of *b* and *k* concurrently. Therefore $2(h-1)$ extra nodes may be locked to handle the overflow and the node-missing problem when ascending the tree path. The value of LN is increased to $h + 1 + 2(h-1)$. The value of LR is increased to 1+2. As for CC, the update operation locks only the leaf node, which must be locked for inserting (deleting) an object, when descending the tree path as [14] does. Following the method stated in [14], an update operation locks at most one node at a time when ascending a tree path. Thus, the values of the three factors of CC are the same as the standard values of the three factors. Finally we compute the difference between the standard summation value and the summation value of the three factors of CC and KB. The difference for KB is $[0+(1/2+(M^h-1)/2(M-1)+(m^{h-1}-1)/(m-1))+1] + [0+((h-1)+2(h-1))+2] = (M^h-1)/2(M-1) + (m^{h-1}-1)/(m-1) + 3h + 1/2$. The difference for CC is [0+0+0] + [0+0+0] = 0. Consequently, CC is better than KB in performance because $(M^h-1)/2(M-1) + (m^{h-1}-1)/(m-1) + 3h + 1/2$ is larger than 0.

5. CONCLUSION

As shown from the analyses of the four CCAs, the analytic model provides deep explanations about the performance of a CCA. This analytic model can formulate the values of the three maim factors, AN, LN, and LR, that influence the performance of CCAs into a standard summation value. We can use the standard summation value to evaluate some proposed CCAs in order to identify which CCA has the best performance. Therefore, an efficient CCA should be designed with these three factors as critical clues. A good designer should design a CCA that makes the values of the three factors as small as possible. Although we use R-trees as illustrative cases, the definition of this analytic model is general and logical. They can be used as guidelines and clues to design an efficient CCA in the areas such as traditional or other spatial access methods.

6. REFERENCES

[1] D. Agrawal, A. El Abbadi, and A. E. Lang, "The Performance of Protocols Based on Locks with Ordered Sharing," *IEEE Trans. on Knowledge and Data Engineering*, vol. 6, no. 5, pp. 805-818, 1994.

[2] R. Agrawal, M. Carey, and M. Livny, "Concurrency Control Performance Modeling:

Alternatives and Implications," *ACM Tran. on Database Systems*, vol. 12, no. 4, pp. 609-664, 1987.

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 322-331, 1990.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing Company, 1987.

[5] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "GENESYS: A System for Efficient Spatial Query Processing," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 519, 1994.

[6] M. Carey and M. Livny, "Parallelism and Concurrency Control Performance in Distributed Database Machines," in *Proc. ACM SIGMOD*, pp. 122-133, 1989.

[7] M. Carey and M. Stonebraker, "The Performance of Concurrency Control Algorithms for Database Management Systems," in *Proc. 10th VLDB Conference*, pp. 107-117, 1984.

[8] J. K. Chen and Y. H. Chin, "A Concurrency Control Algorithm with Efficient Locking on Spatial Data," *proceedings of First International Workshop on Intelligent Multimedia Computing and Networking*, Atlantic City, New Jersey, USA, JCIS'2000, Vol. 2, pp. 726-729, 2000.

[9] J. K. Chen, Y. F. Huang, and Y. H. Chin, "A Study of Concurrent Operations on R-trees," *Information Science*: *An International Journal*, vol. 98, no. 1-4, pp. 263-300, 1997.

[10] J. K. Chen, Y. F. Huang, and Y. H. Chin, "Performance Evaluation of Concurrency Control Algorithms for the R-tree Family," submitted to *IEEE Trans. on Knowledge and Data Engineering*, 1999.

[11] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," in *Proc. ACM SIGMOD Annual Meeting*, pp. 47-57, 1984.

[12] T. Johnson and D. Shasha, "The Performance of Current B-Tree Algorithms," *ACM Trans. on Database Systems*, vol. 18, no. 1, pp. 51-101, 1993.

[13] M. Kornacker and D. Banks, "High-Concurrency Locking in R-Trees," in *Proc. of the 21st VLDB Conference*, pp. 134-145, 1995.

[14] P. L. Lehman and S. B. Yao, "Efficient Locking for Concurrent Operations on B-Trees," *ACM Trans. on Database Systems*, vol. 6, no. 4, pp. 650-670, 1981.

[15] A. Moon and H. Cho, "Performance analysis of global concurrency control algorithms and deadlock resolution strategies in multidatabase systems," in *Proc. of the 6th IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, vol. 2, pp. 653-658, 1997.

[16] V. Ng and T. Kameda, "Concurrent Accesses to R-Trees," *SSD'93, in Lecture Notes in Computer Science (LNCS) 692*, pp. 142-161, 1993.

[17] V. Ng and T. Kameda, "The R-link tree: a recoverable index structure for spatial data," *SSD'93 proc. 5th Int. Conf. on Database and Expert Systems Applications*, in *Lecture Notes in Computer Science (LNCS) 856*, pp. 163-172, 1994.

[18] P. O'Neil, *Database: Principles, Programming, and Performance*, Morgan Kaufmann, 1994.

[19] The Paradise Team, "Paradise: A Database System for GIS Applications," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 485, 1995.

[20] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R$^+$-tree: A Dynamic Index for Multi-dimensional Objects," in *Proc. of the 13th VLDB Conf.*, pp. 507-518, 1987.

[21] V. Srinivasan and M. J. Carey, "Performance of B$^+$-trees Concurrency Control Algorithms," *VLDB Journal*, vol. 2, pp. 361-406, 1993.

[22] M. Stonebraker, L. Rowe, and M. Hirohama, "The Implementation of POSTGRES," *IEEE Trans. on Knowledge and Data Engineering*, vol. 2, no. 1, pp. 125-142, 1990.

[23] A. Thomasian, "Distributed Optimistic Concurrency Control Methods for High Performance Transaction Processing," *IEEE Trans. on Knowledge and Data Engineering*, vol. 10, no. 1, pp. 173-189, 1998.

[24] A. Thomasian, "Concurrency Control - Methods, Performance, and Analysis," *ACM Computing Surveys*, vol. 30, no. 1, pp. 70-119, 1998.

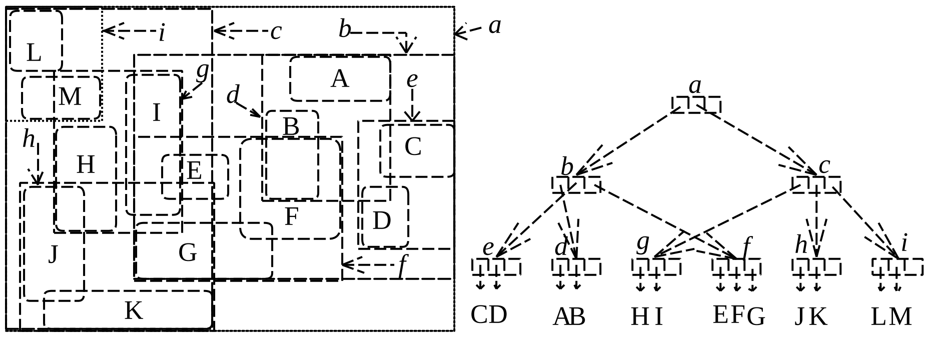

Figure 1. Spatial data objects and the corresponding R-tree.

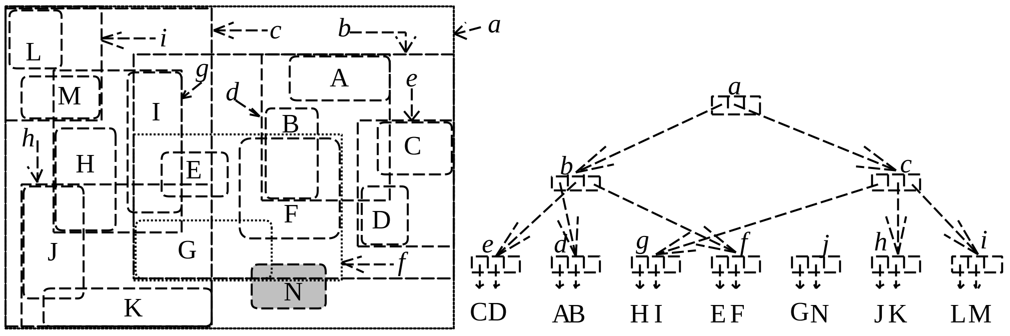

Figure 2. An example for illustrating the overflow-handling.

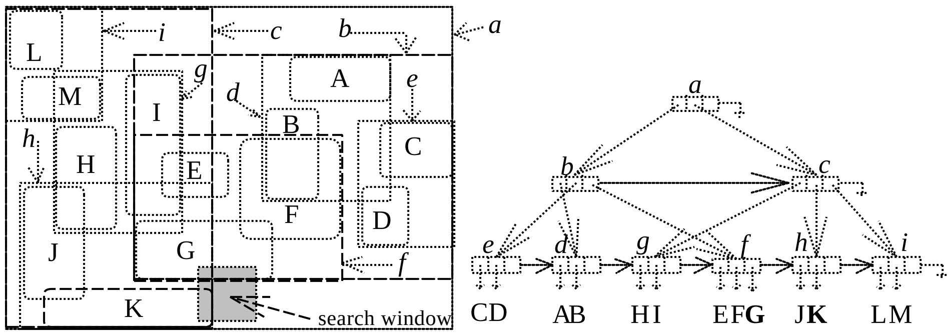

Figure 3. A search example for the lock-coupling CCA.

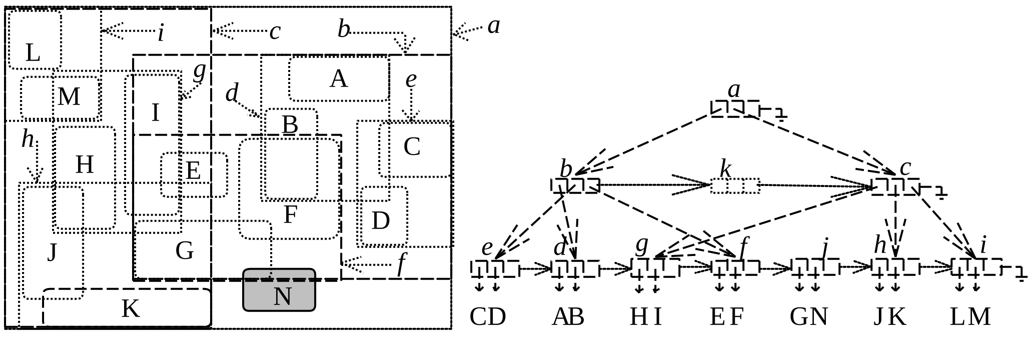

Figure 4. An insertion example for reconstructing the tree.