ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Optimal Search Mechanism using Backtracking in Cloud Environment

K.Govinda*
SCSE,VIT University,Vellore, India
kgovinda@vit.ac.in

Gurunathaprasad V
SCSE,VIT University, Vellore, India
gurunathaprasad.v2010@vit.ac.in

SathishKumar H
SCSE,VIT University, Vellore, India
sathishkumar.h2010@vit.ac.in

Abstract: Cloud computing presents a significant technology trend and is revolutionizing information technology processes and the IT market space. The computing resources are accessed on –demand from pool of configurable computing resources. Accessing data is difficult in mixed and dynamic environment like cloud where resources are accessed and analyzed in real time to address the above issue. This paper presents an optimal search method using backtracking in order to maintain QOS (Quality of Service).

Keywords: Backtracking, Search, Quality of Service, Cloud, Optimal

I. INTRODUCTION

Cloud computing becomes a widespread environment, plenty of data's such as emails, industrial records, defense records etc. had been stored on the cloud in day today life. Since the cloud provides on-demand high quality data storage service the data owners can be freed up from the trouble of the stored data's and privacy of the data's. Although, the data owners and cloud server may not belong to the same trusted domain they put the outsourced data at risk, though the cloud server may not be fully trusted. The priority of the cloud is to encrypt the sensitive data's, in need of data privacy and struggle for unwanted access. Since there are plenty of outsourced data files in the cloud data utilization become a challenging task. Moreover, the data owners may share their outsourced data with as many number of cloud users. In case of individual user, they may want to retrieve a single specific file in which they are interested. By using multi-tenancy or virtualization model the cloud service providers can able to serve multiple cloud users and based on the user demand the physical and virtual resources have been dynamically assigned and reassigned [1]. By using knowledge of the location, formation, and originalities of the resources, the pool-based model concludes that the computing resources become 'invisible' to users. In such a scenario we need an efficient and optimal search over the resources in the cloud as shown in the Fig1.

The best way for selectively retrieve files from the pool is through backtracking approach.

Figure 1. Search in Cloud Environment

II. LITERATURE REVIEW

The American mathematician D. H. Lehmer has discovered the term "backtrack" in 1950. the general back tracking algorithm had been derived from the pioneer string-processing language SNOBOL in the year 1962. Almost all of the computational problem can be solved by general algorithms like Backtracking which formally builds the candidates to the solution in increasing order, and abandons the partial candidate p ("backtracks") as soon as it determines that p cannot be a complete valid solution.[2]. One of the problems which can be solved by applying backtracking algorithm is the eight queen's puzzle, in which the eight queens' should be arranged in such a way that no queens attacks any other. In backtracking approach, the partial candidates are arranged in such a manner that the first k queens in the k rows of the board and all in different rows and columns. A partial solution can be neglected in case they contain mutually attacking queens, since it cannot be a valid solution. The concept of "partial candidate solution" problems can be solved by applying backtracking algorithm and a relatively quick test whether it can be a valid solution[3]. It becomes worthless, for problems such as searching a value in an unordered array. However, backtracking eliminates a huge number of candidates on a single test. It's much faster than a brute force enumeration of all complete candidates.

The constraint satisfaction problems (CSP) such as Sudoku, maze, knapsack and many other can find solution using backtracking.. It becomes a convenient tool for parsing in case of knapsack and other combinational optimization problems[4]. The logic programming languages like Icon, Planner and Prolog are based on backtracking; moreover it's used as a search engine in media wiki software. Backtracking is based on the user-given ""black box procedures" which defines the problem, the character of the partial candidates, and how they are enlarged into a complete candidate. it is a meta-heuristic instead of a specific algorithm although, not like many other meta-heuristic , it assures to find all solution to the finite problem for the given time[5].

A. The Eight Queens Puzzle:

The concept of backtracking is to solve uncomplicated puzzles like eight Queens Problem [6]. The problem is to find a solution for placing eight queens on a chessboard in such a manner that any queen should not attack its neighboring queen. A queen can attack another queen if it exists in the same row, column or diagonal as the queen. Since There are 64 places in a standard chess board is for the first queen, for each of these there are 63 places for the second queen, and so on, for a total of 64*63*….*57 = 178,462,987,637,760 cases. These cases can be reduced by observing that a queen must be placed in a single column. We have 8^8 = 16,777,216 ways of placing eight queens in such a manner that one per column, into eight columns. The solution is a vector of length 8

(a(1), a(2), a(3), ..., a(8)).

a(i), denotes the column for I-th queen where we should place. The solution for 8*8 queen problem is by building the partial solution element by element. We couldn't expand any more, when we reach a partial solution of length k in this case we should back track.

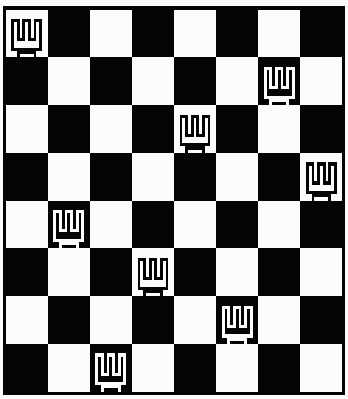


Figure 2. 8 Queens Puzzle

B. The Maze Problem:

In maze the traversing is done from the initial point to the final point.

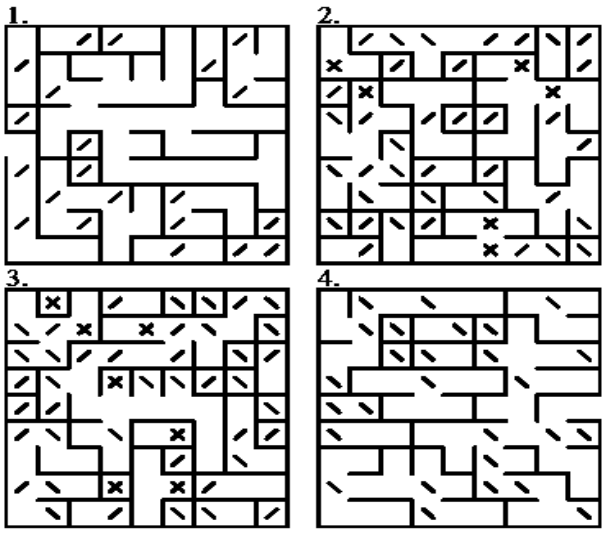


Figure 3. Maze problem

The problem is to choose a exact path for reaching the final point. We should change the direction by backtracking, when it leads to dead end before we reach the final point. We can traverse only in north, south, east and west directions. Until we get to the final point we should move and backtrack. Let us consider a multi-dimensional maze cell [W][H].A wall on the maze is denoted by cell[x][y]=1 and a free cell is denoted by cell[x][y]=0 corresponding to the x, y locations in maze. Initially we want to set the array boundary as 1 for the multi-dimensional array which ensures that we can't get out of the maze and at any time we reside inner part of the maze. Since the array boundary is set to 1 we start traversing from the initial point and to find the upcoming free cell, then the next free cell and the process goes on. Once we got an dead-end then we should backtrack and make assignment as cell[x][y]=1 which denotes a wall on the path as we saw earlier. the process is repeated until we reach the final point[7].

C. The Sudoku Puzzle:

Since Sudoku adapt the concept of "partial candidate solution" it can be solved by backtracking. consider a n*n grid which has subset of n boxes where the grid is filled by the numbers from 1 to n. a normal Sudoku has 27 zones which is classified into rows and columns each of size 9 and 9 squares of 3*3.

The way for solving Sudoku based on backtracking is to construct a graph on n2 vertices, in which each vertex of the grid can be represented in a single vertex. if two boxes belongs to a same zone then an edge is drawn between two vertices. The Sudoku problem is to design the graph with N colors, where the same color should not exists among the adjacent vertices. This can be solved by starting with assigning color to an unfilled vertices based upon some fixed order. While assigning a color to an vertex, we must check whether the assigning color is exist among the filled vertices[8]. If it's false, then we assign the color to the corresponding vertex and the process continuous for rest of the unfilled vertices. Once we try all of the N colors for a single vertex, then we backtrack. We get a solution while all the vertices had been filled with a color. at initial stage, when the boxes of the Sudoku has been filled already, then the back tracking begins after assigning colors and it includes only the empty boxes in the vertex sequence

III. PROPSED METHOD

A. Back Tracking Search:

The backtracking search algorithm computes a group of partial candidates which will give a solution for defined problem. The completion is done incrementally, by a sequence of candidate extension steps[9]. Consider the partial candidates to be the nodes of a tree structure. Every partial candidate is the parent for the candidates which is different in a single extension step. The partial candidate at the leaves of the tree cannot be extended anymore. It traverses this search tree recursively, in such a manner from root to leaves, by following depth-first order. At each node c, it checks whether c leads to a valid solution. If it not, then the sub-tree rooted under c is skipped. Otherwise, the algorithm ensures that c is a valid solution, and it's reported to the user, then it recursively computes all the sub-trees under c. the children's under each node is defined by applying the procedures given by the user.

B. Depth First Search (DFS):

Data structures such as trees, graphs, networks can be traversed by using the Depth-first search (DFS) algorithm. In case of graphs consider any one of the node to be root and traverse along the branch of the corresponding node before backtracking. Initially the vertices are processed in deeper way followed by wider manner. It will recursively process the descendants of the corresponding vertices.

C. DFS Algorithm:

For searching the data inside the cloud we use DFS (depth first search). Consider the data's on the cloud has to be nodes of graph.

Step1. Consider a graph of vertex v and edges e.

Step2. Start at any vertex make the vertex as visited and push the adjacent vertex into the stack until data is found or all the nodes have been visited.

Step3. If the required data is found at the vertex then stop traversing of nodes else search for the process continuous.

Step4. Once the vertex is marked as visited then you could not push the vertex into the stack.

D. Implementation:

Consider the A as the staring vertex and push the adjacent Elements B, D into the stack Then pop the element D at the top of the stack then push the adjacent of D into the stack if the vertex is visited then it should not be pushed .Now pop out the C vertex then push it's adjacent and mark C as visited since the adjacent of C vertex is already inside the stack so no vertex element is pushed into stack. Now pop out the E vertex at the top of the stack then push it's adjacent and mark E as visited since the adjacent of E vertex is already inside the stack so no vertex element is pushed into stack. Now pop out the B vertex at the top of the stack since B is the final vertex of the stack the final Expression of DFS is given as shown in Fig 4.

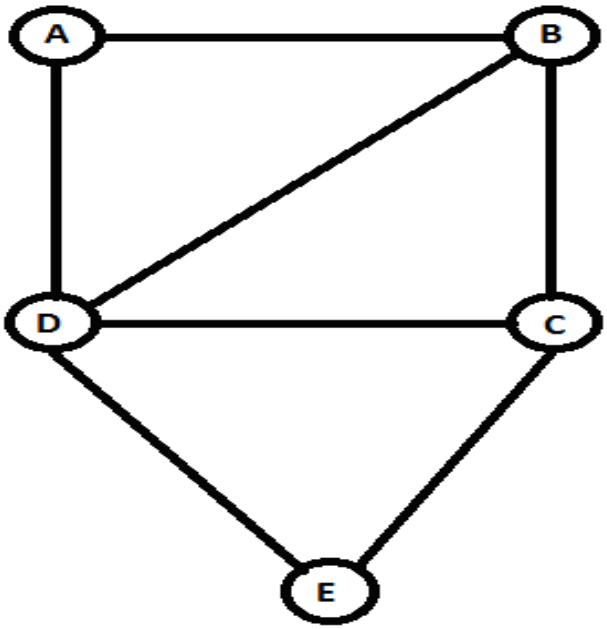
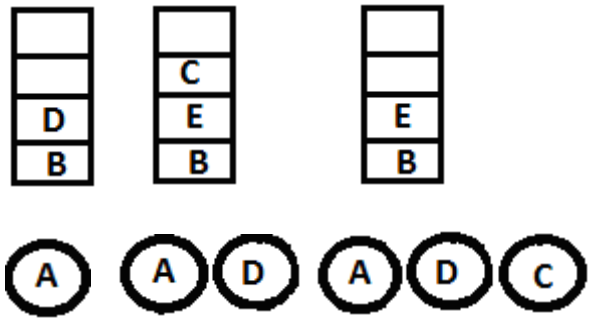


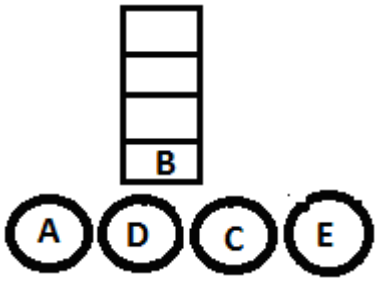
Figure 4. Sample Graph

E. DFS Cconversion:

Step1 **Step2** **Step3**



Step4



Final DFS Expression is :

F. DFS Tree for Graph:

The resultant tree structure after applying DFS on the cloud network is shown in Fig5.

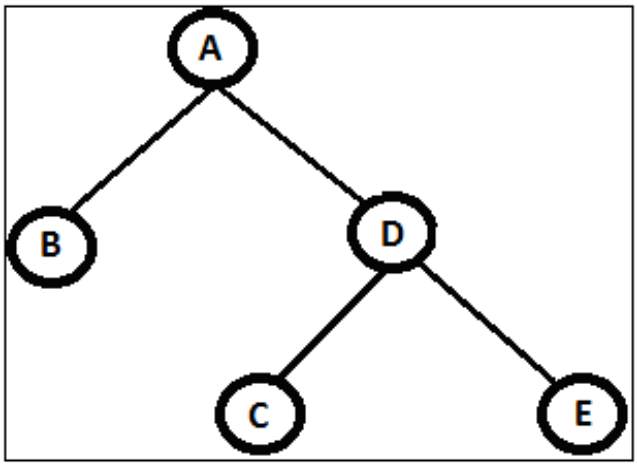


Figure 5. The resultant tree after DFS

IV. CONCLUSION

In this paper we proposed the optimal search mechanism using DFS algorithm in cloud environment. By using DFS the sources in graph can be converted in to tree format which is easier for traversing of sources. More over we can use breadth first search (BFS) algorithm for searching process instead of DFS. This method makes the searching and retrieving of sources faster and reduces the time complexity.

V. REFERENCES

[1] Tharam Dillon, Chen Wu, Elizbeth Chang , " Cloud Computing : Issues and Challenges", In Proceeding of 24th IEEE International Conference on Advanced Information Networking and Applications,2010.

[2] Zhou, R., and Hansen, E. 2004. Breadth-first heuristic search. In Proceedings of the 14th International Conference on Automated Planning and Scheduling, pp. 92–100xzmcnbzxcbvk

[3] Korf, R. 1985. Depth-first iterative deepening: An optimal admissible tree search. Artificial Intelligence,pp.:97–109.

[4] Hohwald, H.; Thayer, I.; and Korf, R.. Comparing best-first search and dynamic programming for optimal multiple sequence alignment. In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), pp. 1239–1245.

[5] Bitner, J.R. and E.M. Reingold (1975), "Backtracking programming techniques," Communications of the ACM, Vol. 18, No. 11, pp. 651-56.

[6] Bernhardsson, B. (1991), "Explicit solutions to the n-queens problems for all n," ACM SIGART Bulletin, Vol. 2, No. 7.

[7] Wikipedia article on Maze Generation Algorithms, http://en.wikipedia.org/wiki/Maze_generation_algorithm.

[8] Korf, R. 1999. Divide-and-conquer bidirectional search: First results. In Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), pp. 1184–1189.

[9] Thomas H. Cormen; Charles E. Leiserson, Ronald R. Rivest, Cliff Stein (1990). Introduction to Algorithms. McGraw-Hill.

[10] J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. Algorithmica, 32(3):437–458, 2002

