

Innovating R Tree for Message Forwarding Technique and Hierarchical Network Clustering in Service Based Routing

Nguyen Thanh Long¹, Nguyen Duc Thuy², Pham Huy Hoang^{3,*}

¹Informatic Center of Hanoi Telecommunications, Hoan Kiem, Hanoi, VietNam

²Post and Telecommunications Institute, Cau Giay, Hanoi, VietNam

³Hanoi University of Science Technology, Hanoi, VietNam

Email address:

Ntlptpm1@yahoo.com (N. T. Long), Nguyenducthuy07@gmail.com (N. D. Thuy), Hoangph@soict.hut.edu.vn (P. H. Hoang)

To cite this article:

Nguyen Thanh Long, Nguyen Duc Thuy, Pham Huy Hoang. Innovating R Tree for Message Forwarding Technique and Hierarchical Network Clustering in Service Based Routing. *Internet of Things and Cloud Computing*. Vol. 3, No. 3, 2015, pp. 59-65.

doi: 10.11648/j.iotcc.s.2015030601.17

Abstract: In service based routing (SBR) [5], the problem for storing forwarding table that includes searching predicates received from subscribers through subscription messages is an important job. When a content request happens, the subscriber will create a subscription message. The subscription message stores several kinds of information, the most important content is a filter that is a conjunction of some constraints [5]. A filter is denoted by F character, that has mathematical formula: $F = \bigwedge_{i=1}^n C_i$, in which C_1 is service request, has the format: $C_1 = \text{'Service_name = requested service name'}$. Every C_i ($i=2..n$) is a constraint that is formed by three components: (Key, op, Value), Key is a keyword for searching, op is an operator, Value is searching condition. Key belongs to the set of name of properties of content messages that the requested service supplies. Op is an operator that depends on the type of data of the Key. Therefore the forwarding table is a set of filters which are received from all subscribers on networks. The algorithms for inserting, updating, deleting and finding filters that match content messages have been published by service providers are very important. In this paper, We mention a technique for forwarding technique on the basic of summary filter for storing and searching filter quickly. This technique is based on some previous researches. In section 8, give an algorithm for finding all network nodes that have matched filters with a content message. Section 7 introduces a cluster routing technique based on summary filter.

Keywords: Service Based Routing, Filter, Constraint, Service, Routing, Tree, Split, Summary, R, Hierarchical, Cluster, Head

1. Introduction

1.1. Subscription/ Un-Subscription Messages

The subscription/ un-subscription message consists of subscriber's address and its content request. In which, content request is a set of some constraints that have a common form: (key, operator, value). A constraint is denoted by C: $C = (\text{Key}, \text{Operator}, \text{Value})$.

In which: Key is a keyword to search content in content message. Operator determines the operation between a pair of a Key and a value. Value is a condition on a property of content message.

So each subscription/ un-subscription message contains a set of constraints that is called a filter that is denoted by F character. Each filter is presented by the formula: $F = \bigwedge_{i=1}^n C_i$, n is number of constraints.

Service based address of a network node is a set of some

filters received from this node that is called a predicate that is denoted by P: $P = \bigvee_{i=1}^n F_i$. n is number of filters. The service based addresses are stored in service based routing tables.

1.2. Service Based Routing Table

The service based routing table stores service based addresses of network nodes. Service based address is extracted from subscription message. These service based addresses are updated regularly by subscription and un-subscription messages: $\text{Routing_Table} = \{P_1, P_2, \dots, P_n\}$.

1.3. Forwarding Technique

Forwarding technique is a searching technique that finds in service based routing table of each service the nodes that have matched subscription messages with each content message received from network.

How to do: i) Standard method by using ternary search tree

and binary search: (a) scan each property of a content message; (b) For each property [p], find all matched constraints with this p; (c) After this loop, get the set of some matched constraints: $\{C_1, C_2, \dots, C_n\}$. (d) Use the *SBR_Couting* algorithms to find all nodes that have made filters that are satisfied; ii) By using summary filter, it will be done by algorithm that is introduced in section 7.

1.4. Summary Filter

Definition:

Assume there are some filters: F_1, F_2, \dots, F_n , the summary filter of these filters is a filter that is denoted by F_S . The summary filter is determined by the formula: $F_S = \bigvee_{i=1}^n F_i$.

So this summary filter satisfies:

$$|F_S| \geq \sum_{i=1}^n |F_i|. \quad (1)$$

Therefore the value domain of F_S covers all value domains of each filter: $F_S \supseteq F_i$; $i = \overline{1..n}$. Operator \supseteq is introduced in the next section.

1.5. Level of a Filter

The number of constraints of a filter is the level of this filter. For example: $F = \bigwedge_{i=1}^n C_i$; $\text{Level}(F) = n$.

2. Some Previous Researches

There some researches on summary filters, that focuses on increasing searching speed and network throughput. There are some researches use summary filters for network clustering [6]. They use summary filter for partitioning subscription messages on some searching nodes. It uses filter summary for maintaing the locality of subscription information. As in [6], they use inaccuracy summary filter to increase network throughput by 200% compares to using accuracy summary filter. In their network, the node automatically adjusts the level of inaccuracy for reducing network congestion. On the other hand, a node can forward subscription messages to other nodes for avoiding repartitioning its filters. R tree have been used in some fields, for example, use R tree for spatial objects searching in Google Map. R tree also been used for storing, searching content based addresses in content based routing protocol.

3. Overview of Summary Filter Technique

3.1. Overarching Concept

Considering the overarching concept, assume that there are two filters F_1 and F_2 are content requirements of one service that arising from network. We suppose that F_1 covers F_2 by the notation:

$$F_1 \supseteq F_2 \quad (2)$$

if all content messages satisfy conditions of F_2 then they satisfy F_1 . Symbolize by:

$$F_1 = \bigwedge_{i=2}^n C_{1i}, F_2 = \bigwedge_{i=2}^m C_{2i} \quad (3)$$

$$\text{In that: } C_{1i} = [K_{1i} \text{ op}_{1i} V_{1i}] \quad \forall i = \overline{2..n}, C_{2j} = [K_{2j} \text{ op}_{2j} V_{2j}] \quad \forall j = \overline{2..m}. \quad (4)$$

For satisfying condition (2) must have: $\{K_{12}, K_{13}, \dots, K_{1n}\} \subseteq \{K_{22}, K_{23}, \dots, K_{2m}\}$.

Suppose the constraint: $P = [K \text{ op } V]$, X is called the value domain of this constraint. If the set of values $\{v\}$ are belonged to X that will satisfy the constraint that means the expression: $\text{Validate}(v \text{ op } V) = \text{true}$.

If the condition (2) is true, at the same time with the conditions: $n \leq m$ and $K_{1i} \equiv K_{2i}$, $\text{op}_{1i} \equiv \text{op}_{2i}$, must have: $X_{1i} \supseteq X_{2i} \quad \forall i = \overline{2..n}$.

That means all the keys and operators are contained in F_1 that are also contained in F_2 , domain value of each of constraint of F_1 covers domain value of each constraint of F_2 with the same key.

We suppose that F_1 and F_2 overlap each other if the following terms are satisfied:

$$X_{12} \cap X_{22} \neq \emptyset, X_{13} \cap X_{23} \neq \emptyset, \dots, X_{1n} \cap X_{2n} \neq \emptyset \quad (5)$$

Assume specific domain of each constraint has magnitude of $|X|$, so that specific domain of the filter has magnitude of:

$$|\text{Domain}(F)| = \prod_{i=1}^n |X_i| \quad (6)$$

3.2. Application of R Tree

When receiving a new subscription message from network interface (I) which requests service S, it has a filter that is denoted by the formula: $F = \bigwedge_{i=1}^n C_i$. We have to check whether F is covered by an existing filter. If there is no filter, have to check whether it overlaps any existing filter. For doing this task efficiently have to apply and innovate R tree, in this chapter it is called R^+ , this tree nowadays is used in many different fields. For example R tree is applied for storing space objects in Google MAP, digital Map, System Paging file with high efficiently in storing and searching.

4. Improving R^+ Tree to Store and Search Filter

4.1. Structure of R^+ Tree

Structure of leaf node: it is a set S that consists of n elements, each element consists of two items P and FS: P is a pointer that points to a filter node F that is indexed in this R^+ tree and FS is a summary filter that covers filter F: $FS \supseteq F$.

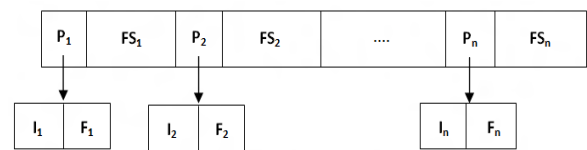


Figure 1. The structure of a leaf node of R^+ tree.

The filters are added to R^+ tree by the algorithm that will be presented in the next section.

- a) Each filter is indexed in R^+ tree that belonged to a filter node that consists of two components: IL and F, in which IL is a list of addresses of routers that have sent subscription messages which have the same filter F, F is a filter that consists of some constraints.
- b) Similar to the leaf node, the structure of an inner node is presented in Fig. 2:

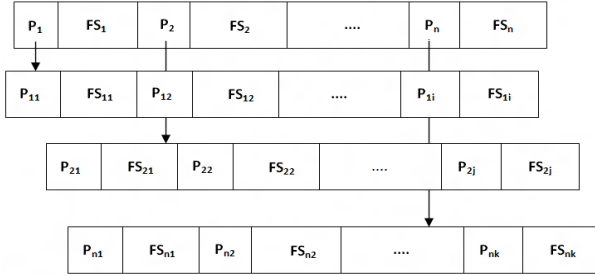


Figure 2. The structure of an inner node in R^+ tree.

- c) Each inner node stores a set of n elements, each element consists of two components: P and FS. P is a pointer that points to an inner or leaf node. FS is a summary filter that covers the whole set of n summary filters of the nodes pointed by P. This means: $FS = \bigvee_{i=1}^n FS_i$. The following diagram describes the covering relationship with $n=6$: Filter of node R covers its children's nodes' filters (R_1 to R_6):

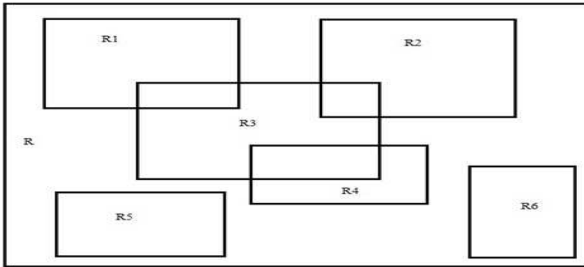


Figure 3. The covering model of R^+ tree.

So We have:

1. Therefore the summary filter of the root node covers the whole forwarding table. So that when checking if a filter belongs to the forwarding table. At first examine whether it is covered by summary filter of the root node. If it is not covered, We conclude this filter doesn't belong to the forwarding table.
2. Each inner node has a number of child nodes which is bound by a lower bound m and an upper bound M : $\text{Amount}_{\text{child}} \in [m, M]$, m and $M \in \mathbb{N}$. Root node and leaf node have number of child nodes which is less than M . Usually We choose $m = \lfloor \frac{M}{2} \rfloor$ or $m = \lfloor \frac{M}{3} \rfloor$ because when number of child nodes of one node reaches $(M+1)$, We have to split this node into two nodes so that each node has an approximately equal number of child nodes. When number of child nodes of a node is less than m , have to remove this node and adjust this tree by an algorithm that will be introduced in the next section.
3. R^+ tree is a balance tree, this means that the height from

any leaf node to the root node of the tree is equal. The proving is based on building tree.

4. Thus the height of the tree from a leaf node to the root node is satisfied the following un-equation system: $\log_M(N) - 1 \leq \text{Height}_{R^+} \leq \log_m(N)$, in which N is number of nodes of tree. Prove: $N \leq \sum_{i=0}^n M^i = \frac{M^{n+1}-1}{M-1}$ and $N \geq \sum_{i=0}^n m^i = \frac{m^{n+1}-1}{m-1} \Rightarrow m^n \leq N \leq M^{n+1} \Rightarrow [\log_M(N) - 1] \leq n \leq \log_m(N)$.

4.2. Establishing and Searching Algorithms

4.2.1. The Establishing Algorithm

At first R^+ tree has only one node, this is both leaf and root of tree, this node points to an indexed filter node that is created to store the filter of a new subscription message that has been received at current router. When adding a new filter F to R^+ tree, We have to find a leaf node N , add F to N . The process for finding node N begins from root node R :

- 1) If R has its child nodes are indexed filter nodes then: $N=R$;
- 2) If R has some child nodes. Assume that R consists of n components: $S = \{C_1, C_2, \dots, C_n\}$, $C_i = (P_i, FS_i)$. For each C_i : i) Find minimum filter F_{\min} satisfies: $(FS_i \vee F_{\min}) \supseteq F$; ii) After doing this loop, will find out $F_{\min \text{ final}} = \min\{F_{\min i}\}$, it is respective to C_{\min} . iii) If there are many C_i satisfy this condition, choose the component that has $|FS_i|$ is minimum.
- 3) Assume C_{\min} has pointer points to its child node R_{child} . Assign temporary node $N_{\text{temp}} = R_{\text{child}}$.
- 4) Apply the step (2) for N_{temp} continuously until N_{temp} is leaf node.
- 5) Assign: $N = N_{\text{temp}}$. Check filter F_i in each indexed filter node (IL_i, F_i) pointed by a component of N . If $F_i \equiv F$, end the algorithm with conclusion: filter F has already existed in this R^+ tree, $IL_i = IL_i \cup \text{Addr}_F$. Otherwise:
- 6) Add new indexed filter node contains F to N .

4.2.2. Searching Algorithm

- a) The searching algorithm finds a leaf node N that has a filter that conforms most to a given filter F . Starting from the root R of tree, assume: $S = \{C_1, C_2, \dots, C_n\}$. Assign R to N_{current} and execute step (b).
- b) For each C_i of N_{current} : $C_i = \{P_i, FS_i\}$, P_i points to Child_i , FS_i is summary filter of Child_i . Check: $FS_i \wedge F \neq \emptyset$. i) If it is true then continue to check. If Child_i is leaf node then execute (c) else assign Child_i to N_{current} and execute (b) recursively; ii) Otherwise, continue (b) with next component.
- c) Calculate: $FS_C = \text{Child}_i.FS \wedge F$. i) If $FS_C \equiv F$: check filter F_i of each indexed filter node of Child_i : if $F_i \equiv F$: end the algorithm with conclusion: F is found in this R^+ tree; ii) Otherwise, denote S_F is a set of some F_i found. Add F_i to S_F . Go to (b) for next component of N_{current} .
- d) Find some maximum items $\{F\}$ from S_F , that belong to some leaf nodes $\{N_{\max}\}$. Thus the set $\{N_{\max}\}$ is result of algorithm.

4.2.3. Algorithm for Adding a Filter to R^+ Tree

This algorithm adds a new filter F to a R^+ tree: check if this filter exists in this R^+ tree. If it's true then add address of node that emitted F to address list of found indexed filter node and exit the algorithm. Otherwise find a leaf node in this tree to store the filter. The algorithm to insert a filter to a leaf node:

- 1) Create a filter node to store filter F and I , I is interface address of the router has emitted the subscription message which contained the filter F . *i)* create a filter node FN , *ii)* $FN.filter=F$; $FN.IL.add(I)$;
- 2) Assume the leaf node for adding F is N . Create new component C_{new} of N , it has two elements P and FS ; Calculate: $N.FS=Summary(F)$; $N.P=FN$. Check: (a) if number of components of N is lower than M then add C_{new} to N , go to step (4). (b) Otherwise go to step (3).
- 3) Have to do: *i)* create new node N_1 , *ii)* split the set of all components of N with C_{new} into two subsets fairly, each of which contains components of one node N or N_1 . Check the condition, whether N has no parent node: (a) If true, new node R' is created, this node has two components C_1 and C_2 . Assign: $C_1.FS=Summary(N)$, $C_1.P=N$, $C_2.FS=Summary(N_1)$ and $C_2.P=N_1$. Now R' is the root of R^+ tree, finish algorithm (this is the proving that the root of R^+ tree has at least two children). (b) Otherwise, go to step (4).
- 4) N has a parent node, this node is denoted by P_N , P_N stores component C_N . C_N has two elements FS and P , P points to N . There are two cases: *i)* if N_1 hasn't been created in (3) then recalculate filter summary for N , assign to FS then go to step (6); *ii)* otherwise go to step (5).
- 5) Create a new component C_{new} of P_N , C_{new} has two elements P , FS . Execute two tasks: *i)* calculate filter summary of N_1 , assign to FS : $C_{new}.FS=Summary(N_1)$; *ii)* assign P to N_1 : $C_{new}.P=N_1$. Go to step (7).
- 6) Check filter summary of P_N , if it has changed, have to assign $N=P_N$, then go to step (4). Continue until N is root node.
- 7) Check number of components of P_N , if this number more than M , assign $N=P_N$, go to step (3). Continue until go to the root node then finish algorithm.

4.2.4. Algorithm for Deleting a Filter From R^+ Tree

Assume need to remove a filter F from R^+ tree.

- 1) Find F from R^+ tree by above searching algorithm. If F is found in the tree and leaf node N contains F , continue next step. Otherwise end algorithm.
- 2) N has n components: $\{C_1, C_2, \dots, C_n\}$, for each component C_i of N (C_i has two elements P_i and FS_i): if $FS_i \wedge F \equiv F$, go to next step.
- 3) Assume F_i is filter of indexed filter node (NF_i) pointed by P_i . There are two case: (i) If $F_i \equiv F$ then remove the address of the node that have made the request from the list of addresses (IL) of NF_i . If this list is empty ($IL \equiv \emptyset$): remove NF_i from this tree. Adjust R^+ by following regulating algorithm, end the algorithm. (ii) otherwise go to step (2).

4.2.5. Algorithm for Regulating R^+ Tree

- 1) Starting from the leaf node S from which a filter is

removed.

- 2) If the number of components of S isn't satisfied the condition (3) then remove S from R^+ , add S to a set Q which stores temporarily all nodes have removed of R^+ and go to step (c). If it is satisfied then finish algorithm.
- 3) If S has parent node then assign $S=S.parent$. There are two cases: *i)* if S is not root of the tree then go to step (2); *ii)* otherwise add root of the tree to Q .
- 4) Get the last node S that has just been put to the set Q . Scan the sub-trees of R^+ tree with root node is S for adding its filters to original R^+ tree by the above adding algorithm.

5. Algorithm for Splitting a Node of R^+ Tree into Two Nodes

5.1. Processing Steps of the Algorithm

When number of components of one node is more than upper bound M , thus having to split this node into two nodes with each node has about $\left\lfloor \frac{M+1}{2} \right\rfloor$ components. Assume current processing node is N :

1. Create new node N_1 , get $\left\lfloor \frac{M+1}{2} \right\rfloor$ components from N to N_1 by one of two algorithms that are described more specific below.
2. If parent of N exists then add N_1 to the set of children of parent node of N and continue step (c); otherwise create new node that will be new root of tree R . Add N and N_1 to the set of children of R , finish the algorithm.
3. Assign $N=N.parent$, if number of child nodes of N is more than M then go to step (1), otherwise finish the algorithm.

5.2. Algorithm Evaluation

The algorithm has maximum complexity in case of regulating tree from one leaf node to the root node of the tree. This is $O[\log_m(N) * K]$, in which N is number nodes of the tree, K is complexity of the splitting algorithm.

6. Algorithm for Splitting Components of a Node into Two Subsets

6.1. Exhaustive Algorithm

6.1.1. Algorithm Specification

Assume number of components of a node is $n=M+1$: $S=\{F_1, F_2, \dots, F_n\}$, choose a random sub-set $S_1=\{F_1, F_2, \dots, F_{\left\lfloor \frac{n}{2} \right\rfloor}\}$ from S . There are: $\frac{n!}{2\left(n-\left\lfloor \frac{n}{2} \right\rfloor\right)!}$ Choices to choose the set S_1 . The remaining elements of S are put to S_2 . For each pair of two sub-sets: calculate F_{S_1} and F_{S_2} are the summary filters for S_1 , S_2 respective. Choose the pair of sub-sets that have: $|F_{S_1} \setminus F_{S_2}|$ is maximum.

6.1.2. Algorithm Evaluation

The exhaustive algorithm has complexity that is in proportion to the number of choices the pair of sub-sets S_1 and S_2 , so it is $\frac{n!}{2^{(n-\lfloor \frac{n}{2} \rfloor)}i}$.

6.2. Quadratic Algorithm

6.2.1. Algorithm Specification

Assume S is a set of summary filters of current processing node that needs to be split: $S = \{F_1, F_2, \dots, F_n\}$ satisfies equation: $n = M + 1$. The requirement of algorithm is: splitting S into two subsets S_1 and S_2 with almost the same size. Require to choose two beginning elements for these two subsets by the following algorithm:

Algorithm 1:

1. Give filter F_{temp} is temporary filter, id_1, id_2 are two integers for storing two indexes of choosen filters, assign filter $F_{temp} = \emptyset$, $id_1 = -1$, $id_2 = -1$;
2. For each filter F_i in the set S of filters execute (3);
3. For each filter F_j in the set S execute (4);
4. Calculate filter F_{ij} that covers both filter F_i and F_j : $F_{ij} = F_i \vee F_j$; calculate: $F'_i = F_{ij} \setminus F_i$, $F'_j = F_{ij} \setminus F_j$, if $|F'_i \setminus F'_j| > |F_{temp}|$ (in which \setminus is filter subtract operator) then assign: $id_1 = i$, $id_2 = j$ and $F_{temp} = |F'_i \setminus F'_j|$.

When this algorithm finished, gain id_1 and id_2 are indexes of two filters ($\{F_{I_1}, F_{I_2}\} = \{F_{id_1}, F_{id_2}\}$) for beginning filters of two sets S_1 and S_2 .

Choose next elements for each set S_i and S_j according to following algorithms:

Algorithm 2:

1. Assume F_1 and F_2 are summary filters of S_1 and S_2 respectively, assign $F_1 = \emptyset$, $F_2 = \emptyset$.
2. For each filter F_i of the set: $S \setminus \{F_{I_1}, F_{I_2}\}$ execute (c).
3. Calculate $F'_1 = (F_1 \vee F_i) \setminus F_i$, $F'_2 = (F_2 \vee F_i) \setminus F_i$, in which denote $(F_1 \vee F_2)$ is filter that is result of extending F_1 to cover F_2 . Go to step (d).
4. If $F'_1 > F'_2$: put F_i into S_2 and assign $F_2 = F_2 \vee F_i$ (4).
5. If $F'_1 < F'_2$: put F_i into S_1 and assign $F_1 = F_1 \vee F_i$ (5).
6. If $F'_1 = F'_2$: put F_i into a set that has the number of elements is less than the other set. If two sets have equal number of elements then put F_i into a random set, recalculate (4) or (5) respectively. Go to (b) for the next item of S .

6.2.2. Evaluating Algorithm

The algorithm 1 has complexity of $O(n^2)$, the algorithm 2 has complexity of $O(n)$. Therefore the complexity of Quadratic algorithm is $O(n^2)$.

6.3. Linear Algorithm

6.3.1. Algorithm Specification

We have to split the set S of summary filters of the current processing node into two approximate equal sets S_1 and S_2 .

Some assumptions:

- a) Each filter of S has a set of predicates, each predicate has

a value domain D .

- b) Each value domain has lower bound D_{min} and upper bound D_{max} , so that $D = [D_{min}, D_{max}]$.

The algorithm for finding two first elements of two subsets S_1 and S_2

- a) For each key K , find maximum lower bound L_{max} and minimum upper bound U_{min} of all predicates of S which have key K . These bounds are belonged to two filters FL_{max} , FU_{min} respectively.
- b) For each K , also find the minimum lower bound L_{min} and maximum upper bound U_{max} of all predicates of S that have key K . These bounds are belonged to two filters FL_{min} and FU_{max} respectively.
- c) Calculate for each key: $V_m = \frac{U_{min} - L_{max}}{U_{max} - L_{min}}$ (6).
- d) Choose two first elements of two subsets S_1 and S_2 from a pair of filters (FL_{min} , FU_{max}) which has V_m that is maximum on all their keys.
- e) With $(n-2)$ remaining filters, put each filter to one of two sets S_1 and S_2 in turn.

6.3.2. Evaluating Algorithm

The complexity of algorithm to choose two first filters of two sets S_1 and S_2 is rated by formula $(|F| * |S|)$, $|F|$ is number of predicates of filter F , $|S|$ is number of filters of S .

The complexity of algorithm to put $(n-2)$ remaining filters to two sets S_1 and S_2 is $O(n)$, n is number of filters of the set of filters S .

7. Cluster Routing Based on Summary Filter

7.1. Cluster routing Concept

Each node builds its own R^+ tree [6] when it receives subscription filter from network. The root of each tree will store filter summary of its routing table. Define a point (P) is centroid of the rectangle that is established from each filter summary. To cluster network based on centroid of subscription filter, assume that: i) the space of all filter is S , ii) S consists of some sub-spaces $\{S_1, S_2, \dots, S_n\}$. So when a node on a cluster receives one filter, it will forward this filter to appropriate cluster head based on point P of this filter.

7.2. Cluster Routing Performance

Use this cluster routing that will reduce time required to find matched subscriptions with each received content message. When a node receives a content message, this content message is a point on space S . It checks to find sub-space S_i that P belongs to and S_i has cluster head (CH_i) respectively. Then the node forwards this content message to CH_i . In CH_i all matched subscriptions will be found using above search algorithm. This cluster algorithm has complexity is $O(n * m)$, n is number of sub-spaces and m is number of constraints of each filter, it is nearly a constant.

8. The Algorithm to Find Nodes Whose Filters Matched a Content Message

Define level of a filter is number of constraints of this filter. In SBR routing protocol, usually uses SBR_Counting to find nodes whose filters matched a content message (M). So the purpose of forwarding algorithm is to find filters that match M. In this section introduce an algorithm to find nodes have filters that match M based on R^+ tree: (i) start from root of R^+ tree, assign N to current node of tree. (ii) Scan each component C_i of N, check if $C_i.FS$ matches M; if it is true then assign $N=C_i.P$, N is node pointed by C_i . If not N is leaf node, continue apply (ii) for N. (iii) Scan each filter F_i of N, check F_i matches M,

if it is true add $F_i.I$ to the set of matched node S: $S = S \cup F_i.I$. Continue algorithm until S contains all network nodes or scan all the matched node of tree.

9. Evaluate Results and Future Development

9.1. The Results of Executing above Mentioned Algorithms

- Build R^+ with number of filters changes from 100000 to 290000, measure the time in ms required to execute for each number of filters to draw the diagram based on these results as below:

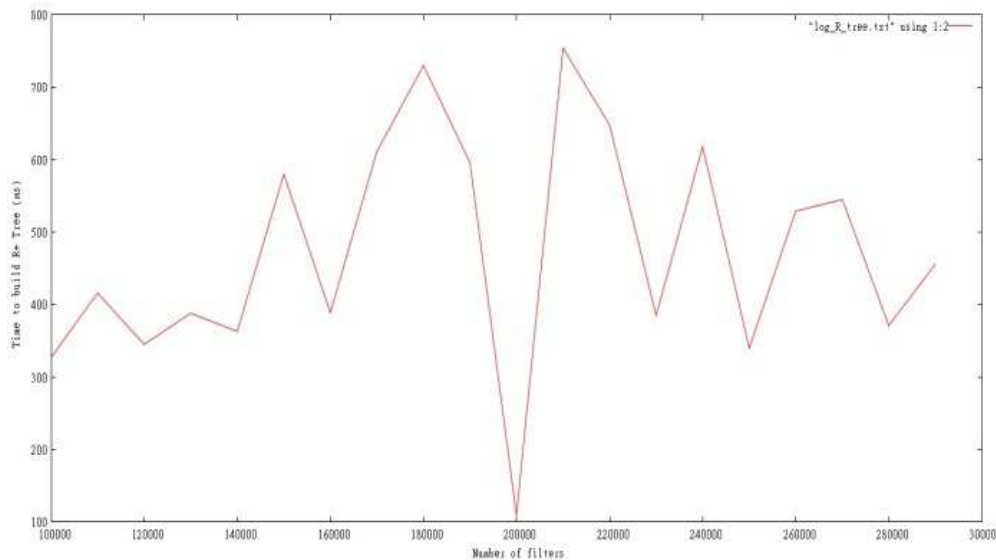


Figure 4. Execution time in ms to build R^+ tree.

As the diagram shows that the time required to build R^+ does not increase continuously when number of filters increases.

- Build a R^+ tree and use the Quadratic algorithm for

splitting node with the number of filters changes from 100000 to 290000 and delete filters, measure the time required in milli second to delete filters and regulate the tree to draw diagram based on the measured results:

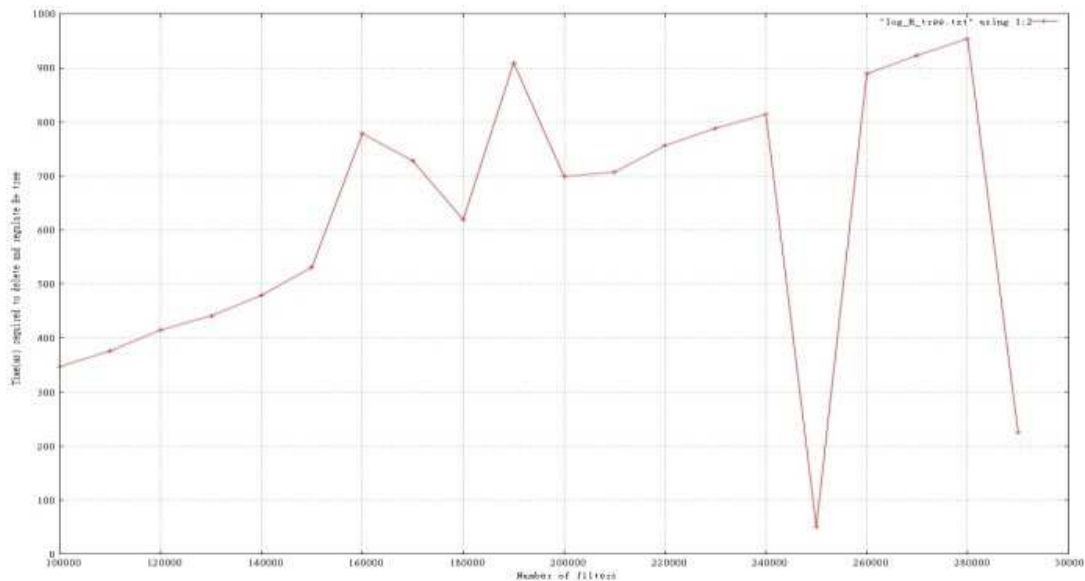


Figure 5. Execution time in ms to delete filters and regulate R^+ tree.

As the diagram shows that the time required to delete filters and regulate R^+ does not increase continuously when number of filters increases.

9.2. Simulation for Applying R^+ Tree for Hierarchical Network Clustering

1. Establish R^+ tree for hierarchical network clustering:
The algorithms for establishing, updating R^+ to hierarchically cluster network as above algorithms for building R^+ to manage service based routing table.
Some modifications: i) A leaf node N_L stores n components (C_1, C_2, \dots, C_n), each component C_i stores a pointer P to an indexed item and a summary filter FS of its child node. Each indexed item stores a pair of two elements node identifier and its filter. Leaf node stores a summary filter of its components and cluster head's ID of its group. So each leaf node give information about one basic group of the network; ii) Similarly, each inner or root node stores information about a upper group of the network. That is a cluster head's ID of its group, some components that have pointers to its child nodes and their summary filters; iii) Condition for inserting a node N to a child group GC of a current group G is: check each components of G , calculate the formula: $FT = |(F \vee F_G) \setminus F|$, F_G is FS of GC , F is the filter of the inserted node. Put N to GC that gives FT is minimum result. Starting this algorithm at root node of R to a leaf node, at the leaf node actually insert N to this basic group and regulate the R tree to the root node for satisfying condition: number of elements of each group in the range $[m, M]$.
2. Simulate these algorithms of applying R^+ tree to cluster network
Execute the process to insert node to the network until there are 1000 nodes. Perform this process by the regulation: when adding enough 10 nodes, then remove 1 node. Measure time in ms needed to add or remove a node, then collect the results to make a below graph:

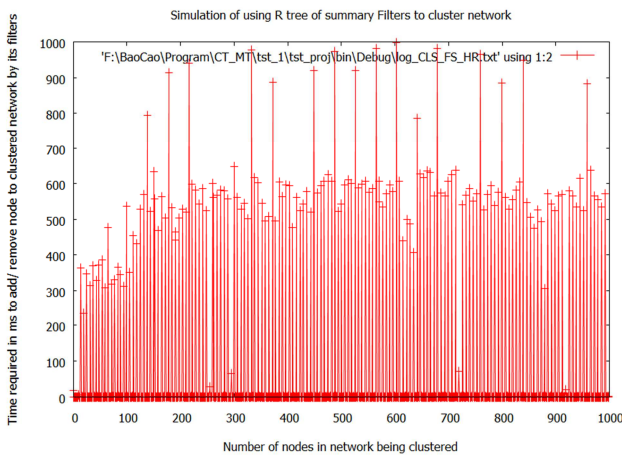


Figure 6. Simulation results of applying R^+ tree to insert/ remove node of clustered network.

As the graph shows that the time required to execute does not increase when the number of nodes increases. It is approximate one milli second. Sometimes this time increases for regulating R^+ tree, but it is lower than one second.

9.3. Future Developing

At the current time above algorithms have been simulated to get above operational results are on numeric value type of predicate, in the future applying above algorithms for value type of string.

References

- [1] J T Robmson, The K-D-B Tree A Search Structure for Large Multidimensional Dynarmc Indexes, 4CM-SIGMOD Conference Proc, April 1981, 10-18.
- [2] Antonin Guttman, R-TREES - A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING, University of California Berkeley.
- [3] Antonin Guttman and StonebrakerM., Using a Relational Database Management System for Computer Added Design Data, IEEE Database Engineering 5, 2 (June 1982).
- [4] G. Yuval, Finding Near Neighbors in k-dimensional Space, Inf Proc Lett 3, 4 (March 1975), 113-114.
- [5] Nguyen Thanh Long, Nguyen Duc Thuy, Pham Huy Hoang, "Research on Innovating, Evaluating and Applying Multicast Routing Technique for Routing messages in Service-oriented Routing", Springer, ISBN: 978-1-936968-65-7, Volume Number 109, 2012.
- [6] Summary-based Routing for Content-based Event Distribution Networks, Yi-Min Wang, Lili Qiu, Chad Verbowski, Dimitris Achlioptas, Gautam Das, and Paul Larson Microsoft Research, Redmond, WA, USA, 200.
- [7] Nguyen Thanh Long, Nguyen Duc Thuy, Pham Huy Hoang, Research on Applying Hierachical Clustered Based Routing Technique Using Artificial Intelligence Algorithms for Quality of Service of Service Based Routing, Internet of Things and Cloud Computing. Special Issue:Quality of Service of Service Based Routing. Vol. 3, No. 6-1, 2015, pp. 1-8. doi: 10.11648/j.iotcc.s.2015030601.11.
- [8] Nguyen Thanh Long, Nguyen Duc Thuy, Pham Huy Hoang, Research on Innovating and Applying Evolutionary Algorithms Based Hierarchical Clustering and Multiple Paths Routing for Guaranteed Quality of Service on Service Based Routing, Internet of Things and Cloud Computing. Special Issue:Quality of Service of Service Based Routing. Vol. 3, No. 6-1, 2015, pp. 9-15. doi: 10.11648/j.iotcc.s.2015030601.12.
- [9] Nguyen Thanh Long, Nguyen Duc Thuy, Pham Huy Hoang, "Innovating R Tree to Create Summary Filter for Message Forwarding Technique in Service-Based Routing", Springer, ISBN: 978-3-642-41773-3, LNICST 121, p. 178, 2013.
- [10] Research on Innovating, Applying Multiple Paths Routing Technique Based on Fuzzy Logic and Genetic Algorithm for Routing Messages in Service - Oriented Routing: Long Thanh Nguyen, Tam Nguyen The, Chien Tran, Thuy Nguyen Duc. Journal: Scalable Information Systems EAI.