International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-4, Issue-ICCIN-2K14, March 2014

Content Based Scheduling of Virtual Machines Using Merkle Tree

Rajat Jangir, Sagar Bajaj, C.M. Sharma, Akshit Pandey

Abstract*— In today's scenario most of the companies are relying on cloud for storing their applications and services. Cloud deployment involves deployment of VM at particular compute node in data centre which requires transfer of VM disk image to compute node. Everyday thousands of VM are deployed in data centres this leads to increase in traffic in data centres. In this paper we proposed a scheme to reduce volume of data being transferred by using Content base scheduling and Merkle tree. In this scheme Merkle tree is used to find content similarity between VMs running on compute node and VM being deployed. The compute node which runs a VM which is most similar is chosen for deployment and hence only dissimilar contents are transferred. This helps in reducing traffic in data centres.

Keywords- Content based scheduling, Merkle tree, Deduplication, SHA1, Content similarity

I. INTRODUCTION

Everyday thousands of VMs are deployed in data centres which lead to increase in traffic in data centres. When a user wants to deploy an application he/she needs to specify instance type and image (operating system).The VM disk image needs to be transferred from storage node to compute node where it can then be instantiated. If all the contents of new VM are transferred this leads to significant increase in network traffic. Hence there is a need to reduce the volume of data being transferred.

Another concern is, when a data centre is down the VMs need to be migrated to some other data centre to ensure that users can access their applications even if data centre on which VM was originally deployed is down. This migration itself can lead to increase in network traffic.

The concept of Content based scheduling can be used along with Merkle tree to reduce the volume of data being transferred. In Content based scheduling we decide the compute node on which VM is to be deployed on the basis of content similarity. Content based scheduling leverages the fact that several VM images are similar, especially VMs with same operating system have high level of content similarity. For eg Content Similarity between Red Hat Enterprise Linux VMs is between 38% and 56% [6].

The main objective of our paper is to select an appropriate compute node for deploying the VM on the basis of content similarity inorder to reduce amount of data being transferred. In this paper we propose a scheme in which Merkle tree is used to find content similarity between Virtual Machines. First, we need to filter out nodes on the basis of available resources and operating system. Then from the filtered nodes we need to find a node which runs a VM which is most similar to VM being deployed.

Manuscript received March 2014
Rajat Jangir, Bhagwan Parshuram Institute of Technology, Delhi India.
Sagar Bajaj, Bhagwan Parshuram Institute of Technology, Delhi India.
C.M. Sharma, Bhagwan Parshuram Institute of Technology, Delhi India.
Akshit Pandey, Bhagwan Parshuram Institute of Technology, Delhi India.

The content similarity is found out by using our algorithm which uses Merkle tree. In our algorithm first Merkle tree for new VM is generated, it is then compared with Merkle trees of VM running on nodes remaining after filtering. Merkle tree is a hash tree which can be used to represent hash values for blocks of a VM disk image. Our Root hash generation algorithm uses SHA-1 hash function to generate hash values. In this algorithm we consider 1 Kb blocks at leaf nodes. Hash Values for these blocks are generated which are then concatenated in pairs to generate combined hash function of the two blocks. At each level Parent node is obtained by concatenating left and right child which are basically hash functions. In this way we can obtain a root hash function for the VM. To find content similarity we compare root hash functions of the VMs being compared then we traverse the tree in left-root-right order to find the branches which are similar. Based on this similarity we can determine the VM which is most similar to the new VM. This traversal and comparison helps us to find the hash functions which are missing in the VM which is most similar to new VM. These missing hash values can be used to find missing blocks in Base VM (VM which is most similar to new VM). These missing blocks can then be transferred to compute node and the required VM can be constructed by using contents transferred and contents from Base VM. So by limiting the volume of data being transferred we can reduce traffic in data centres.

There are other schemes that use content based scheduling to reduce network traffic in data centres but they use Bloom filter[6] for similarity matching. Bloom Filter results in false positive so accuracy of Bloom filter is not very high. Our algorithm does not generate false positive results which ensure that all the dissimilar data is transferred.

This paper consists of following Sections. Section II presents related works and the issue associated with them. Section III describes our proposed model. Section IV reflects the conclusion and finally Section V describes future scope of our work.

II. RELATED WORKS

There are a no of schemes proposed to reduce network traffic in data centres. Some of them are briefly explained in this section

In [1] Keren Jin described effectiveness of deduplication on VM disk Images and proposed use of deduplication to reduce total storage required for VM disk images and allows VMs to share disk blocks. In that paper experiments show factors that effects deduplication between two VM disk Images.

In [2] Xiang Zhang, described how to exploit the concept of data duplication to accelerate live VM migration. The concept of deduplication has been used to reduce the amount of data to be transferred during VM migration. That paper uses hash based fingerprints to find similar and dissimilar memory blocks. That paper is mainly concerned with

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

exploiting deduplication for migration to a specific destination but our scheme is used to schedule VM in such a way that amount of data being transferred is reduced.

In [3] Kazushi Takahashi proposed a Fast Virtual Machine Migration Technique using data deduplication. The technique considers the case in which a VM migrates from one host to another host and then returns back to the original host. The blocks of VM to be migrated are saved at original host prior to migration and when the VM needs to be migrated back at original node only the blocks which were modified at new node needs to be transferred.

In [4] Boris Ederov elucidated Merkle Tree traversal techniques. The creation of Merkle tree is discussed along with various tree traversal techniques. The comparison between various traversal techniques has been done. In [5] Markus Jacobson described Fractal tree representation and traversal and how fractal traversal provides a trade-off between storage and computation. These papers use Merkle tree to verify the contents. However our proposed model uses Merkle/hash tree to compute content similarity.

In [6] Sobir Bazarbayev proposed a similar scheme based on concepts of content based scheduling to reduce the amount of data being transferred. This scheme used Bloom filter for similarity detection. Bloom filter uses k-hash functions to generate fingerprint for a VM. This Scheme was fast as Bloom filter uses Bitwise AND for similarity detection but not accurate enough. In [7] Navendu Jain described the working of Bloom filter and how it can be used to refine Web searches. The use of Bloom filter to determine set membership and similarity matching was discussed. That paper described the fingerprint generation using bloom filter and described the problem of false positive. Sometimes Bloom filter generates false positive results i.e. some content of new VM might not be present in Base VM but Bloom filter still matches the contents. Because of this some contents which are dissimilar between Base VM and new VM might not be transferred to compute node as a result the required VM is not properly constructed at compute node. This is a severe problem as some important contents might not be transferred to compute nodes as similarity has been detected and user might lose some of their valuable information. However our algorithm uses Merkle tree for similarity matching which does not result in false positive. Even though are scheme is slower than Bloom filter it always provides correct results.

III. PROPOSED MODEL

In this section background concepts that were used in our scheme and various algorithms in our scheme have been discussed.

A. Backgrond

The concept of content based scheduling has been used in our paper to reduce the volume of data being transferred. The Scheduling is done in such a way that similar VM are running on same compute node. The concept of data deduplication has been exploited to ensure that amount of data being transferred is reduced provided VM is deployed on a node which runs a VM that is similar.

Propriety Scheduling algorithm are used by most of cloud service providers. So we have used scheduling algorithm used in Open stack [8].This consists of first filtering nodes on basis of basis of available resources (CPU and RAM) then weighing is performed to find most appropriate node to host the VM. we have used content similarity as parameter for weighing.

We have used dedicated node scheduling algorithm [1] in our approach. Dedicated Nodes are the nodes which run VM which have same operating system and this fact can be used to find an appropriate node inorder to reduce traffic. In dedicated Node scheduling algorithm nodes are filtered on basis of available resources as well as operating system. Then from VMs running on nodes remaining after filtering, a VM which is most similar to VM being is found out by comparing the contents using our scheme. The corresponding node is selected i.e. the node on which VM is to be deployed is selected.

Our scheme can be efficiently used to find content similarity and then the node running most similar VM is selected. Then only dissimilar contents can be transferred to compute node.

Bloom Filter can also be used for finding content similarity between VMs [6].Bloom Filter is a m bit vector which uses k hash functions to construct fingerprint [7].The fingerprints of the VMs can then be compared to find content similarity.

1) *Bloom Filter*: It is a m bit vector for generating fingerprints. Bloom filter is also used for checking set membership. Bloom filter uses k hash functions which are applied to each data block. The hash value generated points to a particular location in filter and the corresponding bit is set to 1. This process is repeated for all VM disk blocks, hence fingerprint for entire VM is generated.

An empty bloom filter is shown in fig 1

Fig. 1. Empty Bloom Filter

Here we consider a simple example of a 14 bit bloom filter.

Suppose we have three hash functions hash1, hash2 and hash3. Now suppose there are two string members say apple and plum and we need to generate fingerprint for these two members. Applying hash functions at apple suppose we obtain following : hash1(apple)=3,hash2(apple)=12,hash3(apple)=14.

Fig. 2. Bloom filter showing set bits

In above figure bits corresponding to hash values have been set in bloom filter. Similarly apply hash1, hash2, hash3 to other data member say plum and set corresponding bits in bloom filter. Suppose bits 11, 1, 8 are to be set as shown in figure below

Fig. 3. Bloom filter showing overlapping

Here the bit 11is set for both apple and plum. The reason for this is that size of filter m is very small. The no of hash functions also affect this result.

The problem is that Bloom filter generates false positive results i.e. some contents might not be similar in two VMs being compared but bloom filter approach of matching fingerprints depict that the contents are similar because overlapped mapping. For e.g. a string needs to be searched

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-4, Issue-ICCIN-2K14, March 2014

for in above fingerprint generated. Suppose the string is mango the hash values obtained are 8,12,3. The corresponding bits are already set in above fingerprint so bloom filter depicts that mango is present but the fact is fingerprint was generated only for apple and plum.

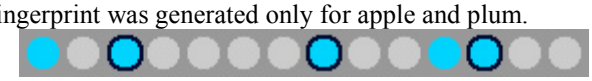

Fig. 4.False positive result showing mango is present in the content

This problem of false positive can lead to incorrect results. All the contents which are dissimilar between the VMs being compared are not transferred and hence the required VM is not properly constructed at the selected compute node .As a result some important contents of user might be lost. Even though is fast but it is not very accurate and compromising accuracy can lead to loss of users valuable information.

B. Our Approach

In our approach we focus at removing the false positive results of bloom filter. The main objective of our study is to use Merkle Tree for matching content similarity and based on which we transfer only dissimilar content between two VMs.

1) Root Hash Generation Algorithm Using Merkle Tree

In our algorithm we aim at providing a way to match the similarity between contents in different virtual machines and then transfer only the dissimilarities reducing the storage space.

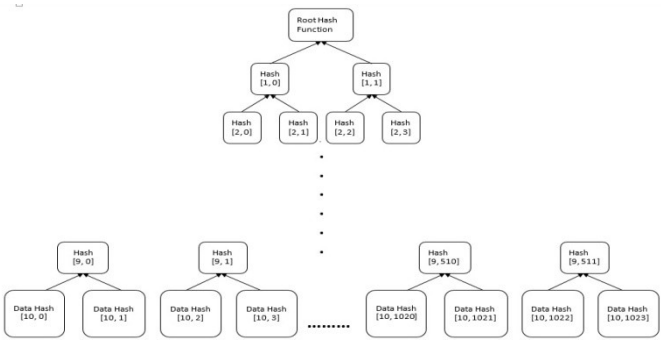

Fig. 5.Root hash generation tree

```
1. we take block size= 1024
2. while block = selectedblock.read (1024) and
                 selectedblock.read is not null do
3. Levels = log2 (block size)
4. j = block size
5.   for k= level->0 do
6.       for (L=0 -->j-1) do
7.          hash[k][L] = GenerateSha1hash [L]
8.          move hash[k][L] to array[k]
9.       end for
10.      for (m=j-1; m>0; m=m-2) do
11.         Noderight = array[k][m]
12.         Nodeleft = array[k][m-1]
13.         if (k=1) do
14.            Nodeparent [k-1][m] = Sha1hash
                    [Noderight||Nodeleft]
15.            go to step 2
16.         end if
17.         if (k>1)
18.            m=m/2;
19.            Nodeparent [k-1][m] = Sha1hash
                    [Nodeleft||Noderight]
20.         end if
21.      end for
22. j=j/2
23. end for
24. go to step 5
```

Fig. 6. Root Hash Generation algorithm using SHA1 and Merkle tree

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

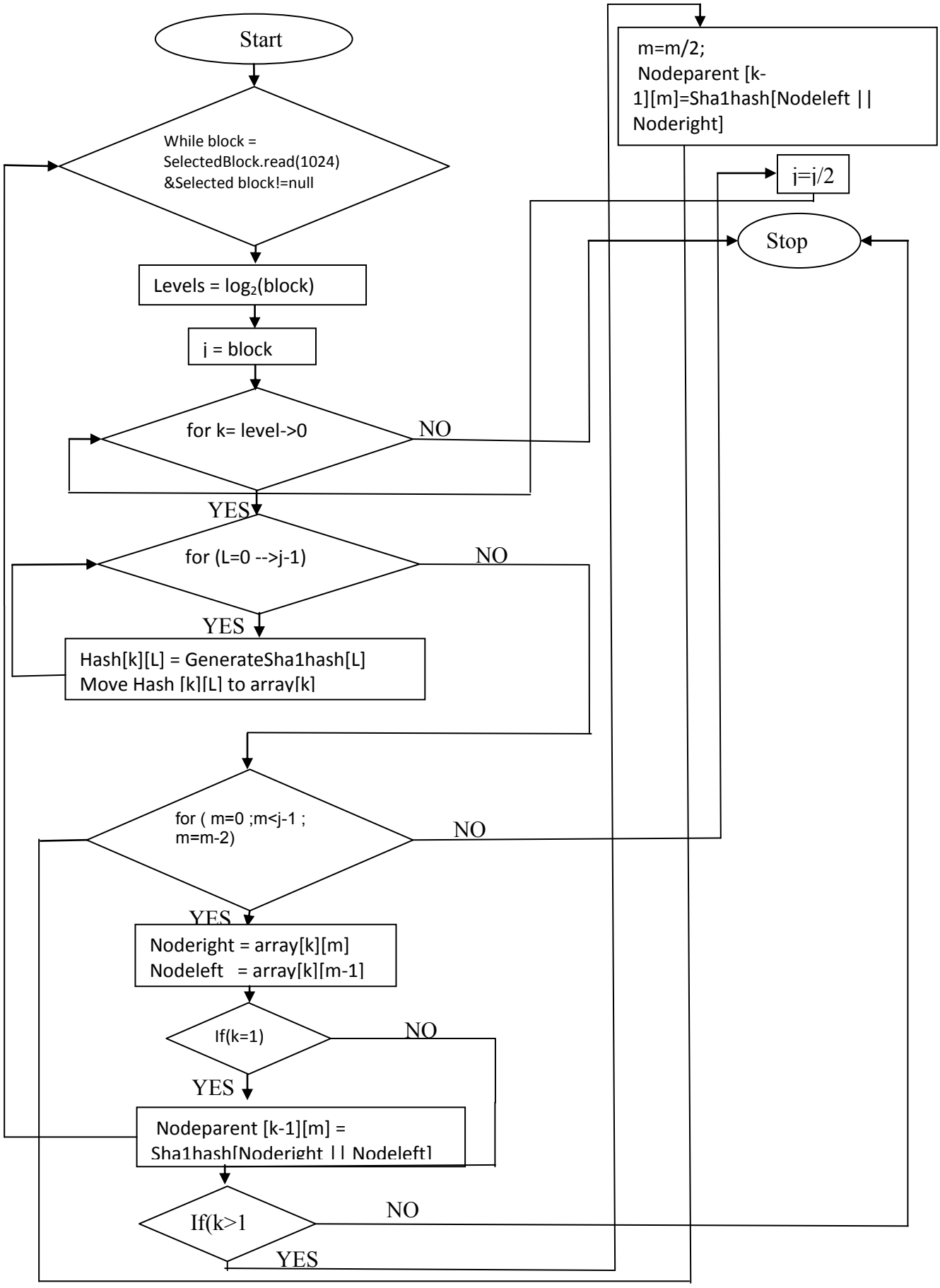

Fig. 7. Flow chart for Root Hash Generation algorithm using SHA1 and Merkle tree

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-4, Issue-ICCIN-2K14, March 2014

The first step of tree generation is that we sort the data and then consider data block of 1 KB as our leaf node and we apply SHA1 hash function on each node. After applying hash function we concatenate two consecutive node and then re-hash the concatenated value. We use a 2-d array for storing the values. In this way we form a 10 level hash tree and have we finally have a root hash function at the top of the tree as shown in fig. 5.

In this algorithm in fig. 6 and fig.7 we take block size of 1024. The while block in step 2 executes till entire data is read. Levels in block define the number of levels of a hash tree in our case it will be eleven. The for loop of k checks the number of elements in an array and iterate for those elements. The for loop of L generates the hash values for same level. The for loop of m is used for concatenating the consecutive node and then storing in the level by decrementing one level. Also we check if we have reached the root node. Value of j is halved every time we make hash values for one level.

Starting from leaf node we move upwards and then we make the root function.

2) Generating a sub-tree based hash tree

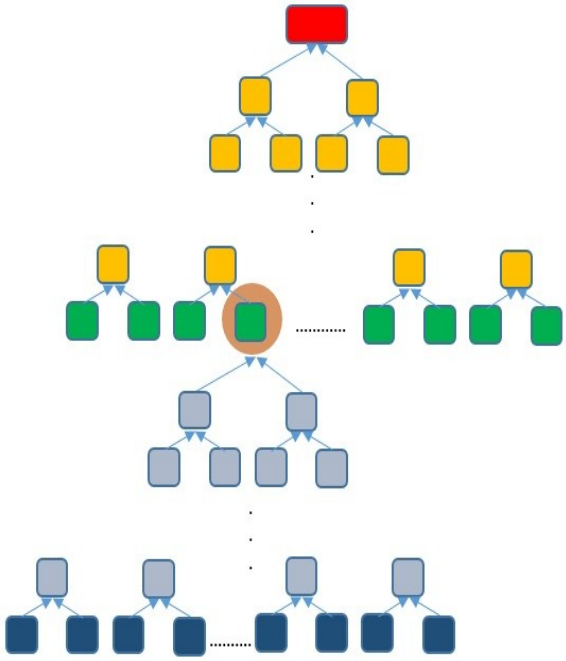

Fig. 8.Construction of sub-tree in a hash tree

With the root hash function algorithm we get a tree that contains 1024 leaf nodes of 1 KB each, which constitutes the data of 1MB. Similarly we can then use the root hash function of 1 MB as a leaf node and construct another tree. This way we have made the root hash function for data of 1 GB. Using the same technique we can construct a hash function for entire data on the disk by making sub-trees as shown in fig. 8.

3) Traversal and comparison algorithm

In our traversal and comparison algorithm we have used the concept of MERKLE fractal traversal algorithm [5]. We start the traversal from root node and see if it matches with the root node of the new VM. If it matches then it means that the data in new VM already exists in the base VM and so we stop the traversal at that particular point and move on to the parent node and then traverse to the right child as show in fig. 9.

1. Compare root hash function of base VM (VM_B) and new VM (VM_N)
2. If (VM_B is not equal to VM_N) then
   - i. Traverse the nodes in order left-root-right
   - ii. Traverse till we reach base node
   - iii. if (base node contains any sub-tree) then
     - a. Go to step 2.i.
   - iv. else we have reached leaf node
3. else
   - i. Stop the traversal downwards as the remaining branch exists in base VM.
4. Backtrack the parent node and see if right node matches.
5. Backtrack similarly till we reach root hash function and we know all branches which are similar and all which are dissimilar and needs to be transferred.

Fig. 9.Construction of sub-tree in a hash tree

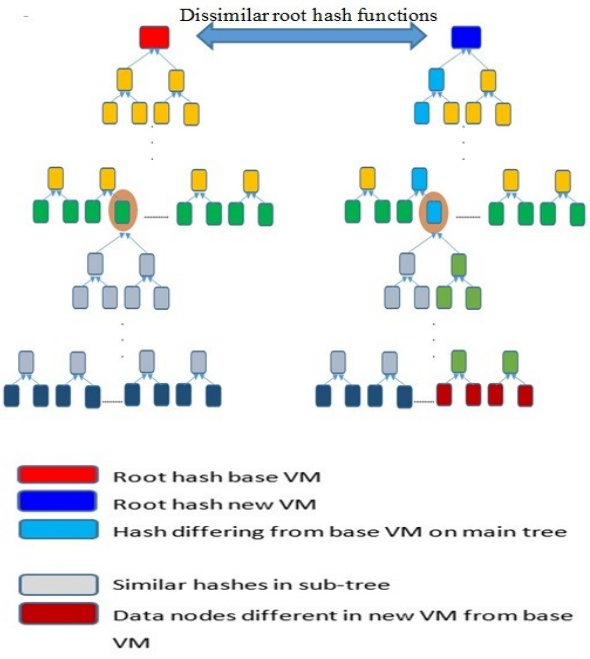

Fig. 10. Content matching between two trees in different VM

As shown in fig. 10 we see that red node represents root node of base VM and blue node represents root node of new VM. As shown we can see the different leaf node with light blue colour that traces down to the brown colour data nodes.

While traversing, all the branches are traversed till we reach to the leaf node of a tree. Once we reach the leaf node we check whether the leaf node have any sub-tree or not. If it has a sub-tree it traverses till it reaches the final data-hash leaf node [4].If we reach the data- hash leaf node then we know that data doesn't exists in the base VM and it needs to be transferred.

Once the entire tree is traversed and all the comparisons are made we compare which of the VMs in the compute node has maximum similarity with our new VM. Once we find the max-similarity VM after hash tree generation and comparison we can transfer data from new VM to base VM. In this way required VM can be constructed by using contents transferred and contents of Base VM.

This approach of content similarity provides very accurate results. Once the entire tree is traversed in this fashion we can determine all the hash values which are missing and the chances of false results is minimum.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

4) Data transfer

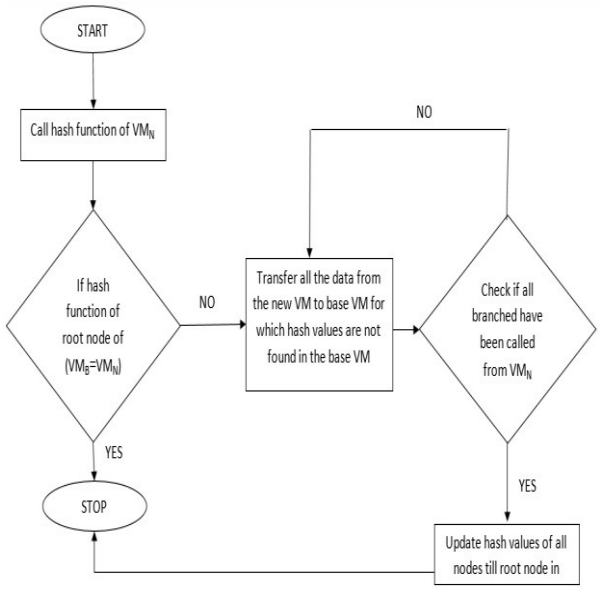

Fig. 11. Data/Content Transfer flow chart

Once we have compared all the VMs then we move onto transferring the data from the new VM to our selected base VM as shown in fig.11. For transferring the data we take the hash values of the data for which we have reached till leaf node while traversing. Then for the respective hash values in the new VM we transfer the data from new VM to base VM.

We check if all braches which were missing in the base VM have been transferred or not and we keep repeating the transfer process till all the data of the branched has been transferred. Once the data is transferred completely we update the new data into our base VM and then we update the hash values and re-construct the hash tree of our base VM.

IV. CONCLUSION

In [6] bloom filter has been used for finding content similarity but it gives false positive results where as our technique provide a more efficient solution for comparing VM. Our scheme does not generate false positive results. Bloom filter is fast technique but is not that accurate. As a result all dissimilar contents between VM being compared might not be transferred and the required VM is not properly constructed at compute node. This leads to loss of users valuable information.

Our algorithm is slower than bloom filter but is more accurate. This ensures that all dissimilar contents are transferred to compute node and required VM can be properly constructed. In our algorithm does not generate false positive results and ensures user data is not lost. Also in future we plan to implement a methodology which can switch between classic and fractal traversal depending on the content similarity hence improve our research.

REFERENCES

[1] K. Jin and E. L. Miller, “The effectiveness of deduplication on virtual machine disk images,” in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conf. New York, NY, USA: ACM, 2009, pp. 7:1–7:12.

[2] X. Zhang, Z. Huo, J.Ma, D. Meng,”Exploiting data deduplication to accelerate live VM migration”, Int. Conf. on cluster computing (CLUSTER) 2010,pp 86-92.

[3] K. Takahashi, K. Sasada, and T. Hirofuchi, “A fast virtual machine storage migration technique using data deduplication”, in Proceedings of 3rd Int. Conf. on Cloud Computing and Virtualization, 2012,pp 57-64.

[4] B. Ederov,“Merkle Tree Traversal Techniques”, BachelorThesis, April 2007, Darmstadt University of Technology, Department of Computer Science Cryptography and Computer Algebra, April 2007.pp. 1-40.

[5] M.Jacobson, T.Leighton, S.Micali, and M.Szydlo,"Fractal Merkle Tree Representation and Traversal" in Proceedings of the RSA Conf. on the cryptographers’ track , 2003 . pp. 314-326. .

[6] S.Bazarbayev, M.Hiltunen ,K.Joshi,W.H. Sanders and R. Schlichting,”Content-Based Scheduling of Virtual Machines (VMs) in the Cloud”, in Proceedings of 33rd Int. Conf. on Distributed Computing System, 2013,pp 93-101.

[7] N. Jain, M. Dahlin, R. Tewari ,"Using Bloom filters to Refine Web Search Results", in Proceedings of 8th Int. workshop on Web and Databases,2005,pp 25-30.

[8] Openstack. Visited on September 5, 2013. [Online]. Available: http://www.openstack.org/

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.