Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

NeuralGDFS: Neural Network Guided DFS for Progressive Cluster Performance on Large Data Set

Suyog Dixit[1] and Dr. Pankaj Dashore[2]

1

Received: 14 December 2014 Accepted: 1 January 2015 Published: 15 January 2015

Abstract

GlusterFS is the most advanced Distributed File System. GlusterFS uses Devis Meyer?s Hashing algorithm as a one way hashing function to save file across the cluster network. Based on GlusterFS, we introduce a new kind of DFS known as NeuralGDFS. NeuralGDFS will incorporate the probabilistic neural network to identify the most probable Brick in which requested file might be located. We also studied GlusterFS performance in virtual cloud environment.

Index terms—

1 Introduction

emands for data storage capacity is increasing day by day. Proprietary DFS leading in storage industry are unable to cope up with the costing, simplicity & customizability as per the requirements of data enterprises.

GlusterFS is developed to solve these problems and extended NAS for the cloud environments.

Gluster allows storage clustering of large number of individual computer nodes to deliver a high performance centrally managed pool. Depending on the workload, capacity and performance can be increased from few terabytes to multiple petabytes. It is possible to connect on-site nodes as well as public cloud infrastructure nodes like Amazon Web Services or Microsoft Azure.

2 a) Gluster Elastic Scaling

Gluster can cluster hard-disk, processor and corresponding input/output resources of multiple thousands of inexpensive computer nodes so that an organization can create a very large storage pool. On demand, if organization wishes to add more storage, they can directly add more hard-disks. If organization needs performance, more levels of virtual RAIDs can be configured by installing more hard-disks distributed among different computer nodes.

Elasticity in Gluster refers to direct proportionality of number of units with the performance. Necessary requirement to achieve elasticity is not to use metadata. Scalable and reliable architecture of fully distributed Gluster server nodes uses parallelism.

There are scenarios with performance throughput exceeding 22GB/s [1].

Author ? ?: Computer Science Department, MITM, Indore, India. e-mails: suyog@suyogdixit.com, dashorepankaj@gmail.com b) No Metadata with the elastic hash algorithm All other distributed file systems keep track of physical allocation of data by indexing file system. But this creates a single point of failure and significant bottlenecks. This also results into logarithmic scaling of the whole system. To avoid this performance chokepoint, Gluster finds location of file using Elastic Hashing algorithm which is based on Davis Meyer's one-way hashing algorithm. Hence, Gluster does not require a metadata sever. With the input of directory name with filename on Davis Meyer algorithm a hash is generated. Davis Meyer's hashing algorithm has determinism and uniformity. Each pathname corresponds unique numerical value.

For the collision of files to the same brick, encryption operations are required (to find same hash value for two files).

Gluster also uses SuperFastHash whose collisions can be easily found eg, finks collides with vinic.

3 How DHT Works

When open() is called, distribute translator is provided with the filename. To determine its location in GlusterFS, translator runs the filename into hashing algorithm. returnctypes.c_uint32(glusterfs.gf_dm_hashfn(filename, len(filename)))

if___name___=="___main___": printhex(gf_dm_hashfn(sys.argv [1]).value)

We installed GlusterFS on Virtualized environment Proxmox 3.0 KVM virtualization on 4 Ubuntu 14.04 LTS instances. We calculated hash as follows:

ganeshji@ubuntu2:~$ python gf_dm_hash.py camelot.blend 0x99d1b6fL

Distribute translator queries to see if it has the mappings for that directory cached. If it doesn't, it looks all the distribute sub-volumes for the DHT mappings for that directory. Let's look on the mappings: The trusted.glusterfs.dht value ends in two uint32 values. By investigating start and end values, most probable location can be found. In our case 0x00000000 <= 0x099d1b6f <= 0x3fffffffe query was sent to brick b. If the file is there, great. That was pretty fast and proficient.

If the file's is not found there, hopefully there's a file there exists a file with the same filename of zero bytes, mode 1000 with the extended feature "trusted.glusterfs.dht.linkto". This is a sticky-pointer, also known as DHT link pointer. This tells the distribute translator that "File is moved to?" This is usually generated while renaming a file. Two network calls will be no big deal If file is not found at brick b, the client calls dht_lookup_everywhere. This sends queries to each distributed sub-volume. In our research setup there is 4x3 volume space, that means 4 queries out of distribute, and 3 queries each out of replicate for a total of 12 lookups. We can understand that this happens in parallel but that's still a lot of overhead.

Imagine a scenario in which we look for files that don't exist repeatedly, this adds a lot of wasted lookups as the client queries every distribute sub-volume every time the file doesn't exist. If that is for example a magneto app, there's commonly a long include path that gets searched for each of 1000 includes. It's common for 30000 non-existent files to be referenced for a single shop home page load.

Quick solutions could be including paths while querying, which practically not possible for each open() request. Next could be a use of metadata server, which is again a problem as discussed in section I[B]. Next solution could be setting a flag for future ease that this file does not exist. But none of the solution is found to be an efficient solution to be problem IV.

4 Neural Network Backed DFS a) Pattern Parallel Training

Pattern parallel training is a method in which full artificial neural network and full set of training data is replicated on each cluster node. This can also speed up the training of the network because nodes broadcasts their computed weights over the network.

5 b) Probabilistic Neural Net

The Probabilistic Neural Network (PNN) model, described by D.F. Specht is a neural implementation of the Parzen windows probability density approximation method, mainly (but not exclusively) oriented toward classification problems. It was originally devised to provide a neural tool capable of very fast training on real-world problems; as compared with the backpropagation, for a given level of performance, the speedup reported was about 200000:1 [2] Neural network have confusion, compression and diffusion properties. We will try to classify hash function based on a neural network. Let us define our neural network as:

Where f i are transfer functions, W i are weights and B i (i=0,1,2) are the bias of the i th neuron layer. As we wish to have a distributed setup, no single point of failure and parallel computation, we take each of the node of Gluster as a node of neural network. The repeated iteration improves the randomness of the relation between H and D and thus strengthens the cryptosystem [3]. Learning of Neural Network will be a continuous process, which also works as a cache system. We also suggested one more layer after output layer as decision layer. This will give final outcome that where a file resides. Figure 2 shows the suggested neural network setup for our NeuralGDFS. A new approach has a fundamental limitation. Let us choose a division of the bricks to store the basic directory structure. It contains directories with allocated GFIDs (GlusterFS ID). This gives "structure" to the network. Files in a given directory is then spread/hashed across many servers. But this is again a direction towards a metadata server.

Solutions like "directory-span" option which limits a given directory's scale-out factor are not effective or ideal because their limits restrict sub-directories' spanning and also impose on the parent directory's spanning.

Rebalance needs to bring the cluster into a balance state by moving data is a time consuming process. While renaming a file, name and cache do not co-reside, causing current design to put the file out of balance, and also is a very state operation triggering probable inconsistencies during its execution. E. Parzen estimated a non-parametric method of KDE (Kernal Density Estimation) for estimation of probability density estimation.

Where, K is kernel positive function, n integrates to one and has mean zero, h is smoothing parameter called bandwidth.

We used KDE.m [5] on MATLAB this example, the data are a synthetic sample of 70 FileIDs drawn from the standard normal and 70 points from a normal distribution with mean 6 and variance 2.

6 NeuralGDFS

In the NeuralGDFS, we keep training probabilistic neural network with random file calls. It is always most probable that a nearby file is called which can be easily computed by probabilistic neural network. MATLAB 2014b was used to simulate the Neuro-DFS environment. No internal tool of MATLAB was used for creation or simulation of neural network.

To increase the efficiency of our neural network, example vector will use cached location of last called file. These are frequently updated as per the misses of neural network. Output layer is not full connected with hidden layer. This is because we wish to have classification of file among the bricks. Hence, node for a given brick is connected only to that brick's output node and not all (See Figure 2). Brick node activation can be calculated via corresponding brick cache product with File as input feature vector. Brick output activation can be then calculated as: Where, N is the total Number of Bricks in cluster space and ? is a smoothing factor. Smoothing factor can be customized while installing NeuralGDFS. If smoothing factor is very large, it will unable to classify a file and if it's too very small, then our Neural Network will not generalize well.

Given with the FileID, the node's brick activations will be calculated and its sum will be forwarded to output layer via classes as bricks. From here, decision layer will take the largest activation will output the brick of FileID.

Almost no training is required hence it is very fast. And as our NeuralGDFS is also based on Network parallel training, rest of the layers also responds very fast. But still as everything depends on Brick's cache, network's consistency problems may arise.

Solution of this is in our architecture of NeuralGDFS. It has tight coupling of whole cluster system with probabilistic neural network. At first when open() is called, we will send our request to DHT module as well as PNN module. If DHT responds first with the corresponding Brick, Cache will updated and then PNN will adjust itself. If there is miss on DHT, PNN's brick will be contacted for file. If still there is a miss, broadcast request to all bricks will be sent for the search. To verify that neural network approximates a function well, we tested MATLAB with own Neural Network. Figure 3 shows number of epochs taken by neural network for learning. Although our PNN will be much faster as it will not be a back propagation neural network. We studied a sample funtion learning on MATLAB.

7 Benchmarking Performance

In our study for Gluster, we used IOzone tool for benchmarking. We applied variety of file operations on our kernel virtualized Ubuntu machines. We used 64 MB Cache surveillance hard-disks and not SSD as most of the industrial environments do.

The persistent rate was less than the burst rate as it does not get advantages of cache or buffer memory in the harddrive.

Read and Write tests were conducted for different sizes with different number of records with number of methods. By comparing even with the two node we can obverse significant increase in the reading and writing speed with respect to a single virtual machine. It can be observed from figure 5 that for similar size of file with similar number of records, higher speed are illustrated in the case of Gluster.

8 Global

9 Brick Classification

In our MATLAB experiment, we have taken FilesID as two element input vector to its associated Brick. Then we converted Brick into its vector form. We then created a new probabilistic neural network. After its testing, its vector output are converted into indices. Then our network classified this new vector with our network. See generated figure 6 which shows how our network classified the FilesID into corresponding brick.

10 Conclusion

Gluster filesystem is a self-integrated, multiprocol, highly scalable upto 72 brontobytes of data supporting infiniband. It can be updated to NeuralGDFS which uses neural network for alternate classification of files among bricks. We suggested procedures which can be developed for better handling of misses of files. This in turn, reduces network lookups and jams. [1] [2] [3] [4]

[1] © 2015 Global Journals Inc. (US)

[2] © 2015 Global Journals Inc. (US) 1

[3] Global Journals Inc. (US)

[4] NeuralGDFS: Neural Network Guided DFS for Progressive Cluster Performance on Large Data Set

1

Figure 1: Figure 1 :

$$FileH_i = Enc_{File_i}(FileH_{i-1}) \oplus FileH_{i-1}$$

Figure 2: DH

$O(2^{\frac{n}{2}})$

Figure 3:

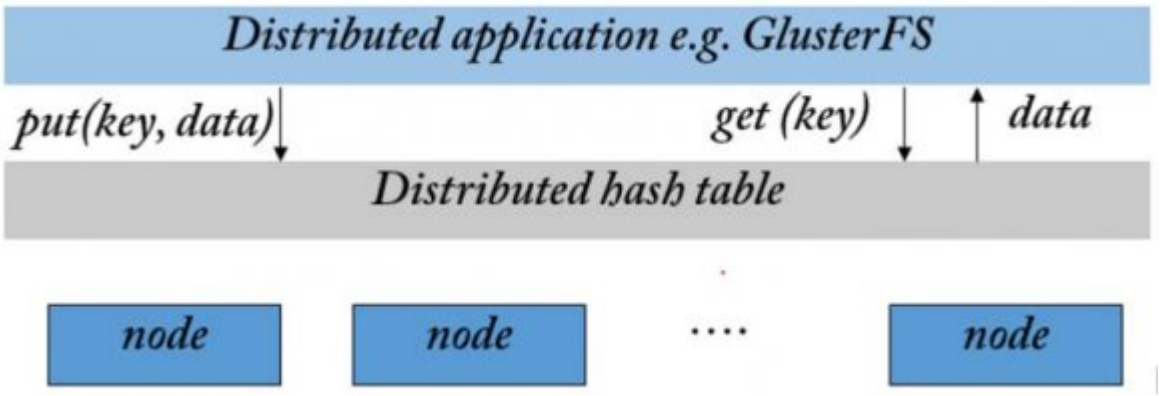


Figure 4: Global

$$_2H = f_2(W_2 f_1(W_1 C + B_1) + B_2)$$

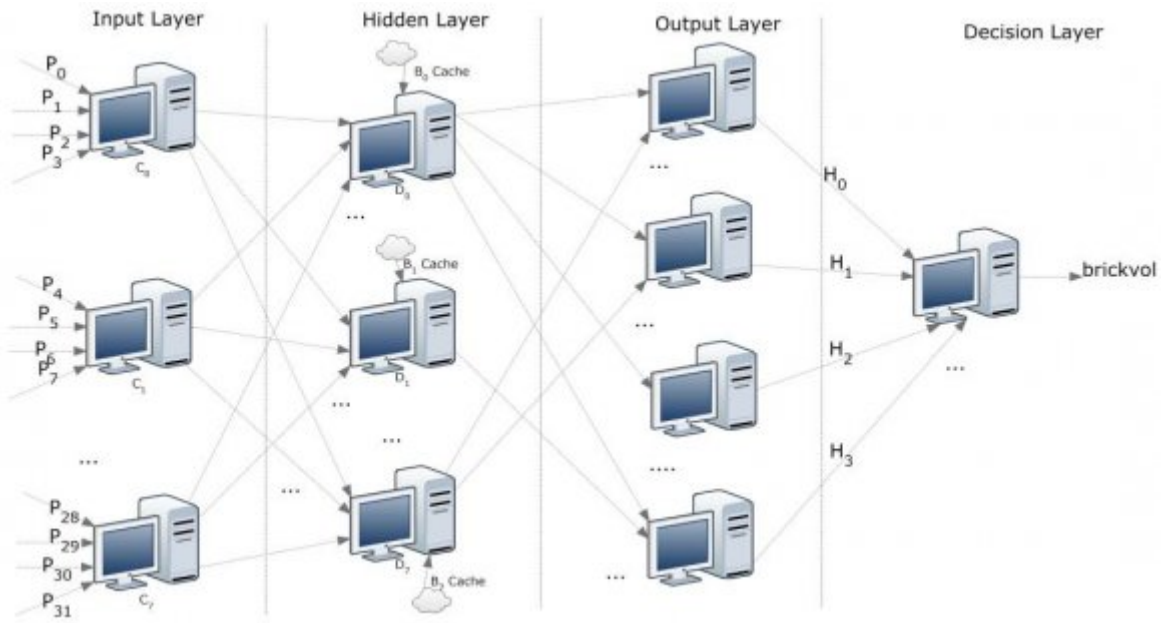
Figure 5: Figure 2 :

$$_{3}H = f_2(W_2D + B_2)$$

Figure 6: Figure 3 :

$$H = f_2(W_2 f_1(W_1 f_0(W_0 P + B_0) + B_1) + B_2)$$

Figure 7:



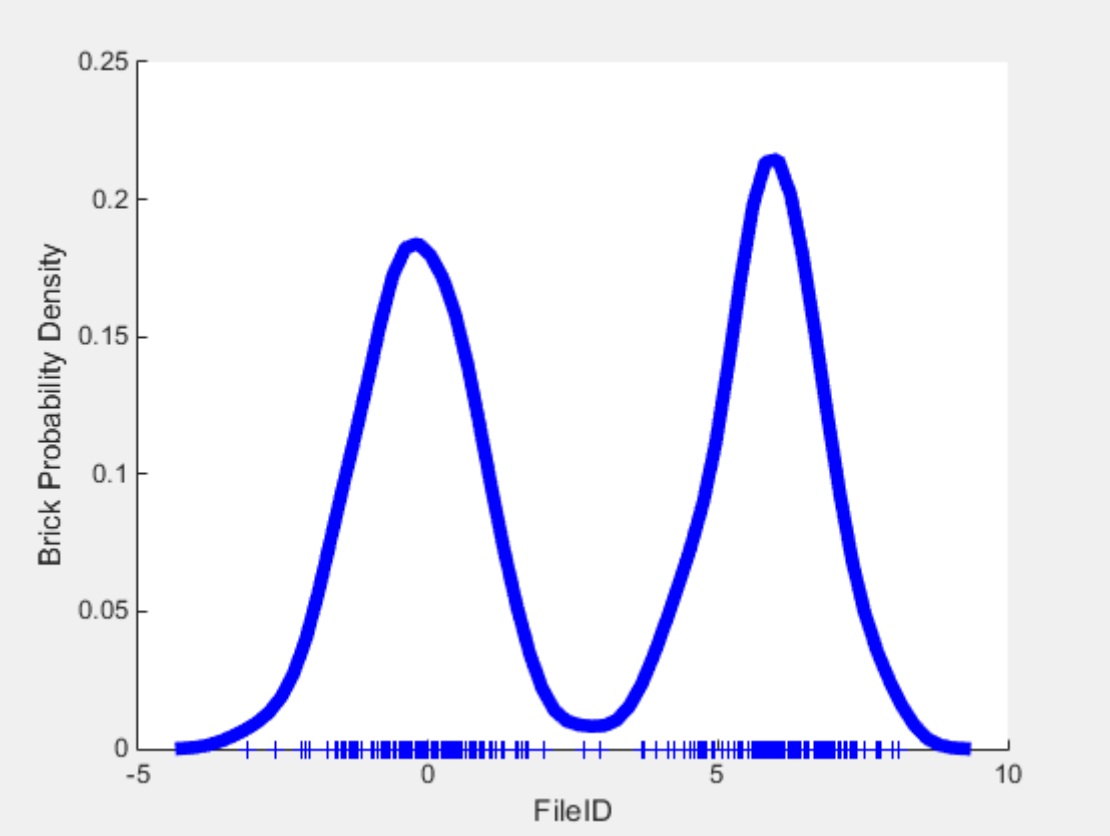
2

Figure 8: Figure 2 :

$$\widehat{Brick}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{FileID - FileID_i}{h})$$

34

Figure 9: Figure 3 :HFigure 4 :



5

Figure 10: Figure 5 :

$$brickvol = \text{argmax}_k \left\{ \gamma^{(k)} \widehat{Brick}_h(x) \right\}$$

6

Figure 11: Figure 6 :

[Gluster ()] *Cloud Storage for the Modern Data Center*, Gluster . 2011.

[Ancona et al. ()] 'Implementing probabilistic neural networks The PNN model'. F Ancona , A M Colla , S Rovetta , R Zunino , EB S . *Neural Comput* 1998. 51.

[Kernel Density Estimator -File Exchange -MATLAB Central (2015)] *Kernel Density Estimator -File Exchange -MATLAB Central*, `http://www.mathworks.com/matlabcentral/fileexchange/14034-kernel-density-estimator` Apr-2015. p. 15.

[Lian et al. ()] *One-way Hash Function Based on Neural Network*, S Lian , J Sun , Z Wang . 2007. p. . (CoRR)

[Dahl et al. ()] 'Parallelizing neural network training for cluster systems'. G Dahl , T Mcavinney , Newhall . *? Comput. Networks, ?* 2008.