Article can be accessed online at http://www.publishingindia.com

Virtual Distributed Disk

Sanjay Kumar*, Pankaj Karki, Deepa Kumari, Amandeep Kaur, G. M. Walunjkar, Subhadeep Ghosh

Abstract

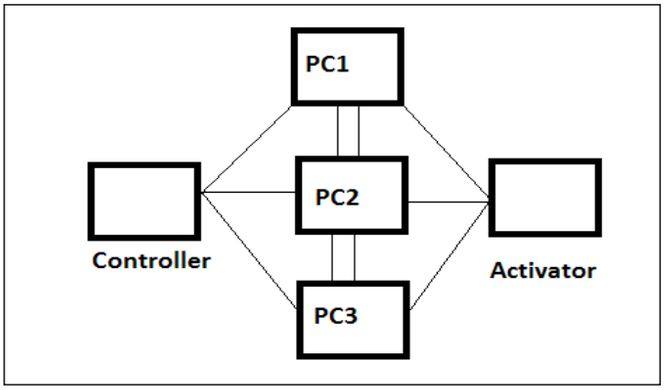
The project is to come up with a mechanism to allow the creation of a Virtual Distributed Disk (VDD). The VDD will be exposed as a device file but will actually be backed by multiple disks residing on multiple machines. Thus, the contents of a VDD can be distributed across multiple machines based on user specified policies. The focus of the project will be as follows: Allow creating, deleting, updating, and reading of any part of VDD. There will be a kernel module to handle various file related system calls and also a system to be able to write to and read from remote sections of the VDD when an application on any machine writes to or reads from it. A caching solution is provided to avoid remote file access as far as possible.

Keywords: VDD, Kernel, Socket, Caching

Introduction

The summary of our idea is shown in Fig. 1 which explains the relationship between the components (components can be pc or laptop) and VDD (Virtual Distributed Disk). The controller share the VDD between the multiple PCs and PCs are connected to each other. These PCs can access the storage of each other.

Fig. 1: Relationships between Components and VDD



In Fig. 1, controller is on the left side which distributes the work to different PCs (PC1, PC2 and PC3). These PCs share a VDD and each PC has disk space equivalent to the sum of the disk spaces of all the PCs. Each PC can access the storage of other PCs. After accessing the storage of each other, PCs will perform some task which is assigned by controller and they will generate some result which is collected by the activator.

Fig. 2 explains how storages are accessed by the PCs and how read and write operations are performed by them. In the figure there are three PCs (PC1, PC2 and PC3) and each PC is having its own cache and storage. When some request comes to a PC then first it checks its own cache. If data is available in its own cache then it performs either read or write according to the request. If the data is not available in the cache then it checks the caches of different PCs where the actual data is located

*Assistant Professor, Government Engineering College, Jagdalpur, Chhattisgarh, India.
E-mail: sanjaydekate19814@gmail.com

Fig. 2: Cache Flowchart for Sharing Storage

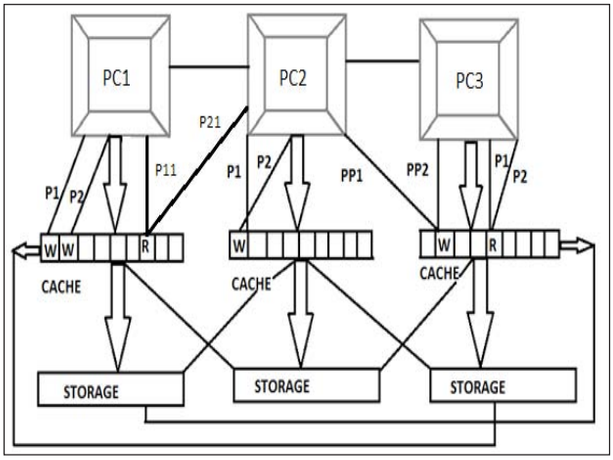


Fig. 2 explains the working of cache of each PC. Each cache is divided into the fixed size of blocks. Multiples reads are allowed on the same block of cache but multiple writes are not allowed on the same block simultaneously. If two processes of same PC try to write the same block then it should not be allowed and this is handled by using semaphore or monitor. In above figure if PP1 process of PC2 and PP2 process of PC3 want to write on the same block of cache of PC3 at the same time then it will not be allowed. Access is allowed only after the first write is over. We can handle this by using Locking Algorithm in which we will lock the block if some write operation is being performed so that other requests of reading or writing that block are kept on a hold.

If two processes of same PC try to read the same block then it should be allowed. For example in Fig. 2, if P1 and P2 processes of same PC (PC3) want to read the same block of PC3 then it will be allowed. If two processes of different PCs try to read the same block of the cache then also it should be allowed. For example in above figure if P11 process of PC1 and P21 process of PC2 want to read the same block of PC1 then it will allowed.

Existing System

Hadoop is a framework which is based on java programming that is used in a distributed computing environment for processing of large datasets. By using Hadoop it is possible to run applications on systems with thousands of nodes that are connected to each other which involve thousands of terabytes of data. Hadoop has one facility which is called distributed file system by using which we can transfer the data between the nodes which are connected to each other. If there is a large number of nodes and they are connected to each other, then in Hadoop we can transfer a large amount of data between multiple nodes through the RAID controller.

New Solution

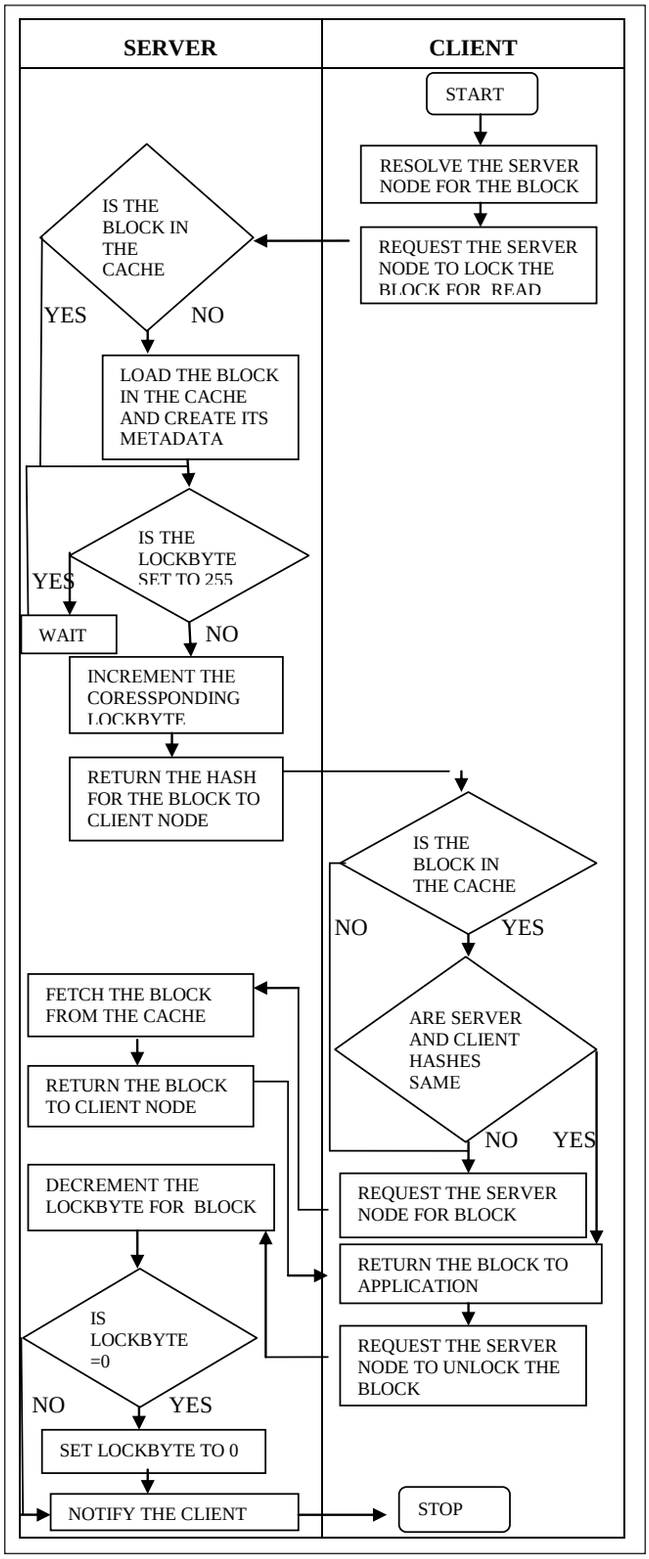
Read Operation in VDD

The client node is the one with the read/write request. The server node is the one where the block to be read/written is present in the storage. Now, when a read request comes to the client, it resolves the address of the server node. The block is checked in cache of the server node. If it is not present, it is loaded in the cache. Then, the corresponding lock byte map is checked. If the value of corresponding lock byte map is 255 that means write operation is currently being performed and access is not granted. In this case, the read request waits until the value changes. If it is anything between 0 and 254 that means read operation is being performed and the value indicates the read count. In this case, the read request is allowed access and the value of the corresponding lock byte map is increased by one as the read count has increased.

Now, after applying the lock the hash for block is sent to the client node. The client node checks whether the block is there in its cache. If it is there it matches the server and client hashes and if same, the block is sent to the application. But if the block is not there in the cache or the hashes do not match, the client node sends a request to the server node for the block. The server node fetches the block and sends it to the client node which returns the block to the application. After the read operation, a request is sent to the server node for decrementing the read count. The read count is decremented in the corresponding lock byte map and the client node is notified. In Fig. 3 it explains how the multiple processes can perform the read operation. One node can behave like as server and client if request is coming from the same node where data is located. If request is coming from the different node and data is stored in different node then first node will behave like as client and second node will behave like as server. If one is currently performing the read operation and another process is coming for read operation then it will be allowed so that the multiple processes can read simultaneously at the same time. We will increment the

corresponding lock byte map. It will return the hash for the block to client node.

Fig 3: Read Operation

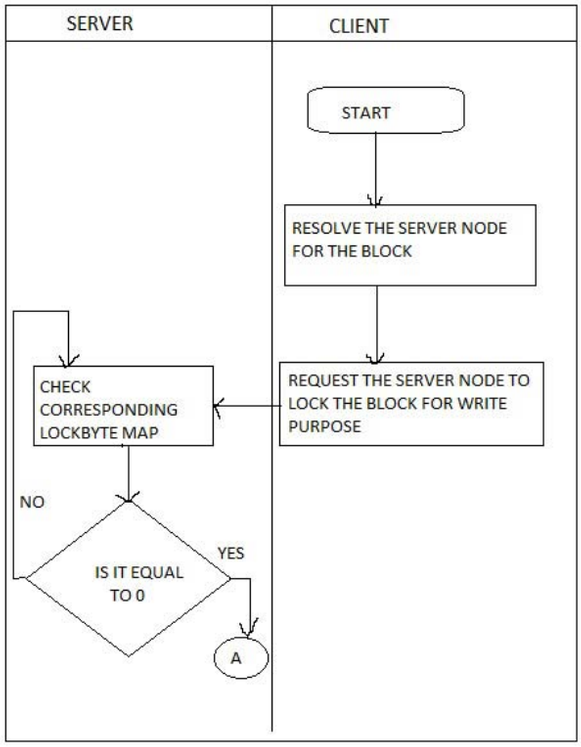


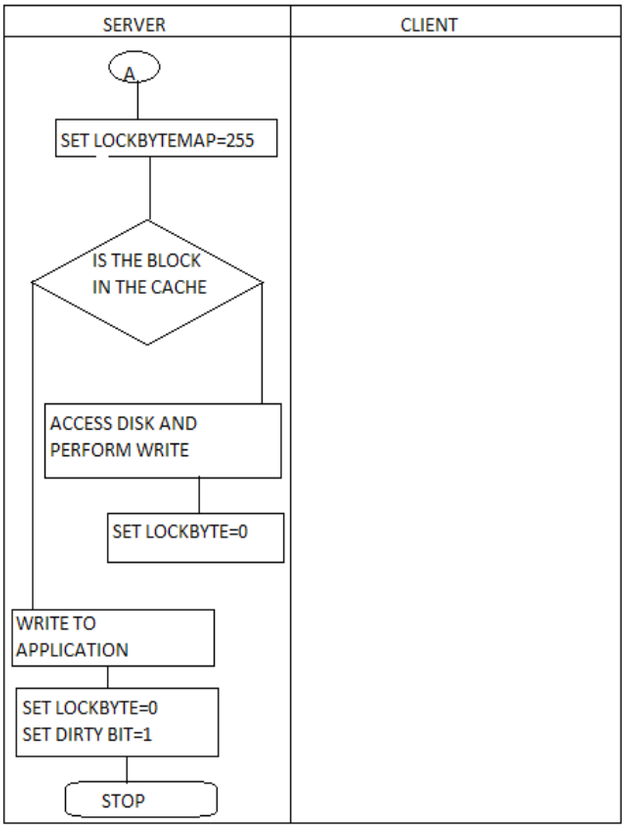
Write Operation in VDD

The basic operation performed in our VDD is read and write operation. Read and write operations are basic operation performed at kernel level.

In our distributed system where each node can access disk of other node virtually, write operation is performed using the concept of server and client. When a write request from comes from a client, it first resolve the server where the block is present after then it request the server node to lock the block for write. The server will look in lock byte map and check if read or write operation is being performed on it. If lock byte map is set from 1 to 254, that means read operation is performing right now then we cannot perform the write operation. Read operation should be performed first and then we can perform the write operation. If no operation is being performed on it then the server node checks its own cache. If it is present in cache it will perform write on it directly. If block is not present in cache it will load the block from disk to cache and then perform write operation. It will then set dirty bit for corresponding block to 1. After regular interval the block with dirty bits=1 are written to disk simultaneously. And the dirty bit is reset.

Fig. 4: Write Operation





Clustering Methods for Connecting Computers

A computer cluster can consist of a set of loosely connected computers or a set of tightly connected computers which will work together. The components of the cluster should be connected to each other which can be connected through the fast local area network (LAN). In a computer system, cluster is set of nodes which can be any pc or server or any resource that will behave like a single system and enable high availability and, in some situations, load balancing and, in some cases, parallel processing.

In most circumstances, all the nodes which are connected to each other can use the same hardware and the same operating system. All the nodes are usually deployed to increase the performance and to increase the availability than a single computer, which will be much more cost-effective than single computers in the terms of speed or availability. The main motive of designing cluster is increasing the performance, but installations will be depend on many other options like fault tolerance also allows for simpler scalability and, in high performance conditions. In our system we plan to use peer-to-peer clustering technique, in which all the nodes will connect to each other through fast LAN.

Algorithms used for Implementing Disk Cache

Least Recently Used (LRU) algorithm is used for replacing the least recently used block from the cache.

We can implement it using AVL tree. AVL trees have certain properties. In an AVL tree, the heights of the two child sub-trees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Also, if we write an in-order traversal, the data are sorted in increasing order, the first value of the traversal being the least value. If we apply this concept, LRU can be implemented in an efficient manner. We can create an AVL tree based on the time at which the blocks arrive. If request for a block which is already there in the AVL tree arrives, then the particular node is deleted from its original position and inserted at a place according to the new time value at which it has arrived. This way we will always have the least recently used value at the first place in inorder traversal of the tree. So, every time we have to replace a block in the cache, we can replace the leftmost node of the tree.

Socket Programming for Interacting with other Devices Contributing to VDD

Socket programming is needed to interact between two pieces of software. We are using socket programming to interact between the different PCs and between the kernel and the user space.

A socket is used to communicate between a client and a server in a given network. A socket can be defined as "the endpoint in a connection." After creation of the sockets, they are used with a set of programming requests or "function calls" which is sometimes called the sockets application programming interface (API).

We are using Unix domain socket for interacting between the processes of same host operating system. If the data is to be exchanged between processes which are within the same OS then these data communication end points are used (UNIX domain socket). This is used for interacting between the kernel and the user space.

If we perform some operation on a socket and if it is unable to complete that operation immediately, it returns the control back to the program. For example, if a read operation is to be performed, some data have to be sent by the remote host. If there is no data waiting to be read, either of the two things can happen: first-the function can wait until some data have been written on the socket. Second, an error can be returned which will indicate that there is no data to be read.

The case in which the function waits until some data is written on the socket is called a blocking socket. We can also say that the program is "blocked" until it satisfies the request for the data. The second case in which an error is returned immediately is the case of a non-blocking socket, and the application should handle the situation appropriately. In our project we will be using non-blocking sockets.

Kernel Programming for Handling Requests within a Computer

Virtual distributed disk separates logical storage from physical storage. It allows access to data irrespective of its physical storage. It hence offers greater flexibility in managing user storage.

Virtualisation allows abstraction of physical location of data and hence achieves location independence. The virtualisation device driver allows data on one device available to all other connected devices. It uses UNIX domain socket to communicate between user application and kernel space. New device can be added with device driver installed on each device.

The data that is being written need to be updated every now and then. We have used concept of 'Lazy Write' to handle this problem. After a fixed interval all the write operation will be written back to storage and a fresh copy will be loaded in cache. When data is written to new space old space is freed and can be reused later. Storage can be assigned where it is needed. The Device driver gives the system an idea that it got a large storage irrespective of its physical location, hence presenting logical view of the total available physical space. The system is able to write to and read from remote sections of the VDD when an application on any machine writes to or reads from it. Kernel programming is used to create the VDD driver. This driver is used to implement the concept of virtualisation of the physical storage. The kernel module allows each node to access all the available storage as if it was its own. Thus enabling each node to read or write to any part of available physical storage. The kernel module makes use of spinlocks for mutual exclusion. Hence take care of consistency of data being read/ written to/ from disk. Multiple storage devices spread across the network act as single monolithic storage device that can be managed independently from each system, hence achieving virtualisation.

Applications

Virtual distributed disk can find applications in various fields. Wherever big data is involved, VDD can be used there. Nowadays, applications need to be fast. All companies want that the processing time should be minimum for faster access and interpretation of data. VDD caches the recently used data that is accessed from the disk. It creates a disk cache so that the next time when some previous data is needed, it is loaded from the cache and not from the disk. It avoids the latency of accessing the data from the disk and thus performs many operations in a limited time.

So, Virtual distributed disk can be used in big data management companies like IT and banking sectors where a large amount of data needs to be processed or accessed.

Conclusion

Hadoop is an open-source software framework for storage and large-scale processing of datasets on clusters of commodity hardware and uses RAID controller for sharing the large data between the nodes. But in our project no RAID controller is used for sharing the large data. In this project PCs directly access the storage of each other through disk cache which makes it faster and reduces latency for sharing the storage of each other.

If a new device (computer) is to be added to the network of virtual distributed disk, connections need to be established between existing and the new device. When this happens, the entire work needs to be halted for some time so that the new device can be added successfully. Also, if some computer goes down, a backup should be there so that the data can be recovered. These things need to be worked upon in future.

References

Anderson, T. E., Dahlin, M. D., Neefe, J. M., Patterson, D. A., Roselli, D. S., & Wang, R. Y. (1996). Server less network file systems. *ACM Transactions on Computer Systems*, February 14(1), 41-79.

Anderson, T. E., Owicki, S. S., Saxe, J. B., & Thacker, C. P. (1993). Highspeed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, November, 11(4), 319-352.

Bialecki, A., Cafarella, M., Cutting, D., & Omalley, O. (2005). *Hadoop: A framework for running applications on large clusters built of commodity hardware*. Retrieved from http://lucene.apache.org/hadoop.

Birrell, A. D., & Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, February, 2(1), 39-59.

Cabrera, L. F., & Darrel, D. E. (1991). Long. Swift: Using distributed disk striping to provide high I/O data rates. *ACM Computing Systems*, Fall, 4, 405-436.

Cao, P., Lim, S. B., Venkataraman, S., & Wilkes, J. (1994). The Ticker TAIP parallel RAID architecture. *ACM Transaction on Computer System*, 12(3).

Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., & Stoica, I. (2010). *Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling.* In Proceedings of the 5th European Conference on Computer Systems, 265-278.

Retrieved from http://en.wikipedia.org/