Efficient Data Prefetching Approach for PC-Cluster based Cloud Storage System

Tin Tin Yee
University of Computer Studies, Yangon
tintinyee.tty@gmail.com

Thinn Thu Naing
University of Computer Studies, Yangon
thinnthu@gmail.com

Abstract

Cloud storage system architecture and design plays a vital role in the cloud computing infrastructure in order to improve the storage capacity as well as cost effectiveness. To address this need, the cost effective PC cluster based storage server is configured to be activated for large amount of data to provide cloud users and is implemented with Hadoop Distributed File System (HDFS). HDFS is open source distributed file system that designed on low cost hardware. In this system, high access latency occurs due to the access mechanism of HDFS. Data prefetching is an effective technique for improving file access performance which can reduce response time delay for I/O system. In order to solve this issue, we propose data perfetching algorithm based on FP-growth algorithm to extract user access patterns from user's historical accesses records for reducing the communications between clients and NameNode. According to user's access frequent patterns, frequently access data are stored in client cache.

Keywords: *Cloud Computing, Cloud storage, Data Prefetching*

1. Introduction

Cloud computing is a consequence of economic, commercial, cultural and technologies conditions that have combined to cause a disruptive shift in information technology towards a service-based economy. Cloud computing offers more opportunity to creative and ambitious people by lowering up-front costs to start new businesses. Many cloud computing service providers have been continuously made efforts to cut down expenses of system maintenance by developing energy-efficient and cost-effective infrastructure and platform software. In addition to the technologies reducing the maintenance costs, it is necessary to reduce a significant up-front investment to build the cloud computing service. Cloud computing applications require scalable, elastic and fault tolerant storage system.

The storage demands of cloud computing have been growing exponentially year after year. Rather than relying on traditional central large storage arrays, the storage system for cloud computing consolidates large numbers of distributed commodity computers into a single storage pool, and provides a large capacity and high performance storage service in an unreliable and dynamic network environment at low cost. To build such a cloud storage system, an increasing number of companies and academic institutions have started to rely on the Hadoop Distributed File System (HDFS). HDFS provides reliable storage and high throughput access to application data. It is suitable for applications that have large data sets, typically the Map/Reduce programming framework for data-intensive computing. HDFS has been widely used and become a common storage appliance for cloud computing.

The rest of this paper is organized as follows. Section 2 describes background and related work. In section 3 discusses PC cluster based cloud storage system architecture. In section 4 describes proposed data prefetching algorithm and evaluation of the proposed architecture presents in section 5. Finally, section 6 is the conclusions.

2. Background and Related Works

PC-cluster based cloud storage system is designed for reliably storing very large files across distributed commodity machine in a large cluster [7]. It stores each file as a sequence of block; default block size is 64MB and all blocks in a file are the same size except the last one. The blocks of a file are replicated for high availability and fault tolerance. The replication policy is random replica placement policy that replicas of a block are placed at random on any of the machines in the entire cluster. The default replica factor for single data block is three.

Young el al. [8] presented an intelligent prefetching technique that significantly improves hybrid- flash-disk storage which is a combination of hard disk and flash memory. They proposed two-level prefetching technique and it considered both file level and block level prefetching. This algorithm has good performance on disk perfetching. However, it doesn't have a high efficiency at prefetching data in cloud environment. Yong el al. [1] proposed an Algorithm-level Feedback-controlled Adaptive (AFA) to address data access delay.

Lin el al. [6] proposed an Affinity-based Metadata Prefetching scheme to provide aggressive metadata Prefetching. Heung el al. [5] proposed the Double Predication-by-Partial-Match Scheme (DPS) that can be used under modern web framework and proposed the Adaptive Rate Controller (ARC) to determine the prefetch rate depending on the memory status dynamically.

James el al. [2] presented graph techniques for prefetching to reduce file system latency. They create a graph with nodes as files in the file system and edges to represent the access sequence between nodes. Peng el al. [3] proposed a weighted graph based prefetching algorithm for metadata servers in petabyte-scale storage systems. In this approach, relationship graphs are constructed dynamically by defining successor relationships.

Jiazheng el al. [4] proposed a Real-time Data Prefetching algorithm based on sequential pattern mining to hidden the data access delay. This has a good performance under a reasonable prefetching size threshold. They showed that this algorithm has a high hit rate with low prefetching rate but the increased network communication is little. The proposed algorithm directly prefetching data objects from remote data nodes to local data nodes cache space and hasn't consider the replacement and consistency strategy.

3. PC-Cluster based Cloud Storage System Architecture

In this section describes physical topology of PC-cluster based cloud storage system. The system architecture is shown in figure 1. The overall framework of PC-cluster based cloud storage system consists of three layers that are web based application services layer, Hadoop Distributed File System (HDFS) layer and PC cluster layer.

The web based application service layer provides interface for the users whose can store and access their own applications such Virtual Machine (VM) images, data files and multimedia data, etc. The HDFS layer supports the file system for PC cluster layer. HDFS is as a user-level file system in cluster which exploits the native file system on each node to store and access data. The input data are divided into blocks, typically 64 MB and each block is stored as a separate file in the local file system. HDFS is implemented by two services: the NameNode and DataNode. The NameNode is responsible for maintaining the HDFS directory tree, and is a centralized service in the cluster operating on a single node. Clients contact the NameNode in order to perform common file system operations, such as open, close, rename, and delete. The NameNode does not store HDFS data itself, but rather maintains a mapping between HDFS file name, a list of blocks in the file and the DataNode(s) on which those blocks are stored. The PC cluster layer provides to store large amount of data. This layer tries to transfer from the cluster computing to storage system and uses inexpensive PC components. The large files can be stored by striping the data across multiple nodes. In PC cluster layer consists of one NameNode and many DataNodes. It uses HDFS (Hadoop Distributed File System) to store and access data in the collection of the nodes. Each node has its own memory, I/O devices and operating system. The nodes are physically separated and connected via a LAN.

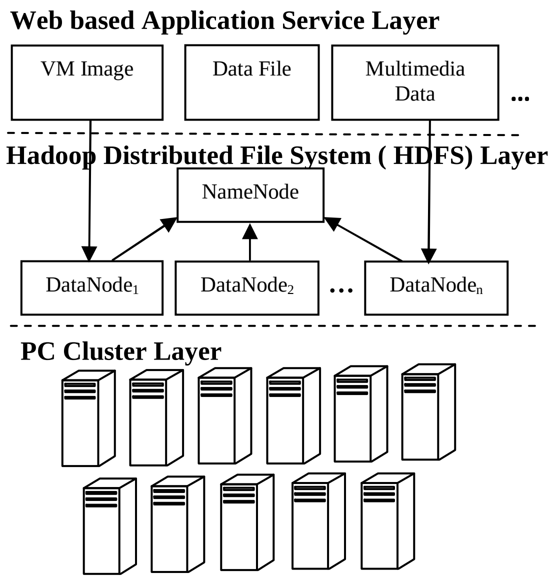

Figure 1: PC-Cluster based Cloud Storage System Architecture

4. Proposed Algorithm for Data Prefetching

In this section presents proposed data access operation of PC-cluster based cloud storage system and proposed algorithm to improve access performance of this storage system. Prefetching is an effective approach for improving data access performance which can reduce access latency for I/O systems. In the cloud storage system, prefetching for user frequently access files is critical for the overall system performance. In this paper presents data perfetching algorithm based on FP-growth algorithm to extract user access patterns from user's historical access log records.

4.1. Proposed Data access operation of PC-cluster based Cloud Storage System

When cloud users read file from PC-cluster based storage server, they first interact with NameNode for file metadata and then perform actual I/O operations directly with relevant DataNodes to retrieve data. High access latencies occur due to the access mechanism of HDFS. Firstly, users need to query NameNode for file metadata which happens once for each file access. Secondly, sequential files are not placed sequentially in block level, or even are placed on different DataNodes although its own data placement strategy. Finally, HDFS currently doesn't provide prefetching function to hide I/O latency. In this case, response time delay caused by reading from NameNode that is the major performance bottleneck of processing user request.

To solve the performance bottleneck, in this paper proposes prefetching and caching function using FP-growth algorithm to extract user access patterns from user's historical accesses records for reducing the communications between clients and NameNode and frequently access data are stored in client cache using Least Recently Used (LRU) replacement policy to improve performance of PC-cluster based storage server for cloud storage.

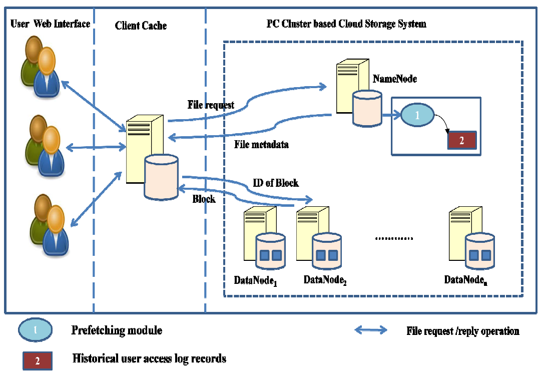

Figure 2: Proposed data access operation

The proposed data access operation of PC-cluster based cloud storage system is shown in figure 2. In this operation, Users frequently access data are prefetched from NameNode's historical access log records using FP- growth algorithm. According to user's access frequent patterns, frequently access data are stored on client cache using Least-Recently-Used (LRU) replacement Policy.

In LRU, every *data* in cache has a time-stamp assigned when inserted or when found in cache. It selects candidates for removal at cache finding the oldest files in the cache using the time-stamp stored in the cache with the *data*.

4.2. Proposed Data Prefetching algorithm

In this section describes proposed prefetching algorithm and defines definitions as follows:

Definition 1: D$_{cache}$ represents user request data exist in client cache.

Definition 2: D$_{no\text{-}cache}$ represents user request data doesn't exist in client cache.

Definition 3: Access$_{log}$ represents historical user access log records.

Let m ={m$_1$, m$_2$,..., m$_i$} be a set of items in historical access log, and a transaction in access log file M={M$_1$, M$_2$,...,M$_n$}, where n is the total number of historical access log. The support (or occurrence frequency) of a pattern A, where A is a set of items, is the number of transactions containing A in access log. The pattern A is frequent if A's support is no less than a predefined minimum support threshold, ξ. Table 1 and 2 list the summary of notations used in the algorithms.

Table 1. Notations Used in the FP-tree Algorithm

Notation	Description
M	The set of access log
ξ	The predefined minimum support threshold
m	The set of frequent items
L	The list of frequent items
p	The first element of sorted frequent-item list
P	The remaining frequent-item list

Algorithm 1: Data Prefetching Algorithm

```
Begin
   1.  Let user request be R
   2.  Let client cache be C
   3.  for user request R arrive in data
       access operation process do
   4.  if  R in D_cache then
   5.    Return user request R
   6.  end if
   7.  else
   8.    Call prefetching module(Access_log)
   9.    Return user request R
   10.   Store R in C with LRU replacement
       policy
   11. end for
End
```

Procedure prefetching module (Access$_{log}$)

```
Begin
   1. Find user frequently access patterns
      from Access_log using FP-tree and
      FP-growth algorithm
   2. Fetch data from DataNodes

End
```

Algorithm 2: FP-tree Algorithm

```
Begin
 1. Scan the M.
 2. Collect m, the set of frequent items, and
    the support of each frequent item.
 3. Sort m in support-descending order as L,
    the list of frequent items.
 4. Create the root of an FP-tree, T, and label
    it as "null".
 5. For each transaction Trans in M do
    begin
 6.     Select the frequent items in M
 7.     Sort them according to the order of L.
    end
 8. [p |P] select from Sorted frequent-item list
 9. Call insert_tree([p | P], T ).
End
```

Table 2. Notations Used in the FP-growth Algorithm

Notation	Description
P	The single prefix-path part of Tree
β	The combination of the nodes in the path P
	The itemset in the access

	log records
a_i	A frequent item in the tree

Algorithm 3: FP-growth Algorithm

```
Begin
    1.  if (Tree contains a single path P )
    2.   then for each combination of the nodes $\beta$ in the path P
    3.   generate pattern $\beta \cup \alpha$ with support = minimum support of nodes in $\beta$
    4.  else if for each $a_i$ in the header of Tree{
    5.  generate pattern $\beta = a_i \cup \alpha$ with support = $a_i$.support
    6.  contruct $\beta'$s conditional pattern base
    7.    then $\beta's$ conditional FP_tree Tree $\beta$
    8.  if Tree $\beta \neq \emptyset$ then
    9.
    10.      end if
    11.    end if
    12.  end if
End
```

Procedure insert_tree([p | P], T)

```
Begin
    1.  if (T has a child $N$ and item-name = p.item-name)
    2.  then increment N's count by 1;
    3.  else create a new node $N$, with its count
    4.   initialized to 1,its parent link to T
  end if
End
```

5. Performance Evaluations

In this section presents theoretical analysis of both original data access operation and proposed data access operation. Assume that there are N files in the PC-cluster based cloud storage system, whose lengths are denoted as $L_1, L_2, \ldots L_n$.

Definition 4: When access data from PC-cluster based cloud storage system using original data access operation, the access operation is
composed of following steps:
Step1. User send a read request to NameNode.
Step2. NameNode looks up the metadata of request files.
Step3. The metadata is returned to user.
Step4. User sends a read operation to a relevant DataNode.
Step5. DataNode fetches the requested block from disk .
Step6. Data block is returned to user.

Analysis: Assume that step 1 and 4 are constant time δ because they are consumed by sending commands and step 3 is assumed that constant time δ because the size of metadata is very small. Step 2 is β times because NameNode need to look up once for each file access. Step 5 is α times where M stands for the number of blocks. DataNodes fetches the total number of blocks for read request. Step 6 is $R_{block} = \frac{l}{speed}$ times which depend on length of files l and network transmission speed $speed$. The total access time $Total_{access}$ is defined as

$$Total_{access} = (3 * \delta) + \sum_{i=1}^{N} \beta + \sum_{i=1}^{M} \alpha + \sum_{i=1}^{M} R_{block} \quad (1)$$

Definition 5: When access data from PC-cluster based cloud storage system using proposed data access operation, the access operation has two cases.
Case 1: User request data exist in Client cache, the access operation is composed of following steps:
Step1. User sends a read request to Client cache.
Step2. Client cache looks up the data of request files.
Step3. The data is returned to user.

Analysis: Assume that step 1 is constant time δ because it is consumed by sending commands. Step 2 is $\sum_{i=1}^{N} \beta$. Step 3 is $R_{block} = \frac{l}{speed}$ times.
The total access time T_{access_cache} is defined as

$$T_{access\_cache} = \delta + \sum_{i=1}^{N} \beta + \sum_{i=1}^{M} R_{block} \quad (2)$$

Case 2: User request data doesn't exist in Client cache, the access operation is processed as following:
Step1. User sends a read request to Client cache.
Step2. Client cache looks up the data of request files.
Step3. User request data doesn't exist in Client cache, prefetching module find user frequently access patterns from $\text{Access}_{\text{log}}$.
Step4. Data block is returned to user and client cache.

Analysis: Assume that step 1 is constant time δ because it is consumed by sending commands. Step 2 is $\sum_{i=1}^{N} \beta$. Step 3 is $Prefetch_{time} = \sigma \sum_{i=1}^{M} \alpha$ which find user frequently access pattern and fetch request data from relevant DataNode. Step 4 is $R_{block} = \frac{l}{speed}$ times which depend on length of files l and network transmission speed $speed$. The total access time $T_{prefetch\_access}$ is defined as

$$T_{prefetch\_access} = \delta + \sum_{i=1}^{N} \beta + Prefetch_{time} + \sum_{i=1}^{M} R_{block} \quad (3)$$

In numerical analysis, we assume that δ = 0.128ms, β = 5 ms, α = 5 ms and R_{block}= 0.64ms. The results of numerical analysis are shown in figure 3. According to the results, in the proposed approach, data access time is depending on client cache. Therefore, the proposed algorithm is more accuracy increase as well as the data access time is also fast.

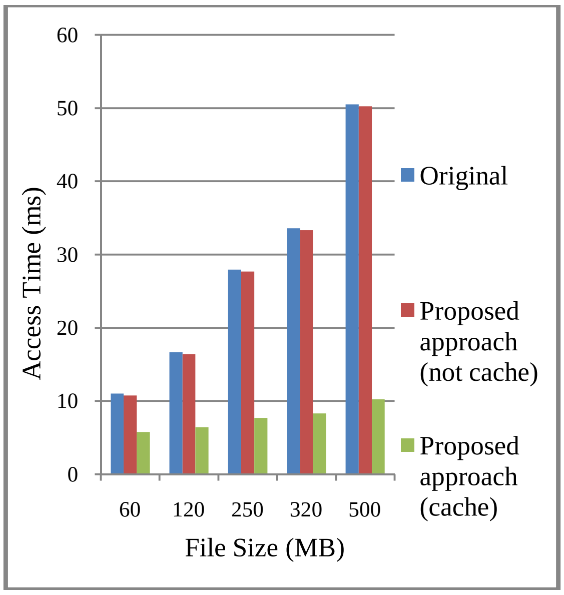

Figure 3: Numerical analysis of proposed methods

6. Conclusions

In this paper proposed data prefetching and caching function using FP-growth algorithm to extract user access patterns from user's historical accesses records for reducing the communications between clients and NameNode and frequently access data are stored in client cache using Least Recently Used (LRU) replacement policy to improve performance of PC-cluster based storage server for cloud storage. According to performance evaluation, the access performance is improved when more frequently user access data are stored in client cache but cache storage space will be large and more time consuming process. As future works, we plan to investigate more effective solutions to detect and predict data prefetching method for improve I/O performance of PC-cluster based cloud storage system.

References

[1] C.Yong, Z. Huaiyu and S.Xina-He, "An Adaptive Data Prefetcher for High-Performance Proccessors", *In Proceedings of the Tenth IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2002, pages 155-164.

[2] G. James and A. KY Randy, "Reducing File System Latency using a Predictive Approach", *In Proceedings of the USENIX Summer Technical Conference*, 1994.

[3]G. Peng, W.Jung, Z.Yifeng, J.Hong and S.Pengju, "Nexus: A Novel Weight-Graph-Based Prefetching Algorithm for Metadata Servers in Petabyte-Scale Storage Systems", *In Proceedings of Sixth IEEE International Symposium on Cluster computing and the Grid*, 2006.

[4] L.Jiazheng, W.Shaochun, G. Yunwen and Y.Bowen "Real-Time Data Prefetching Algorithm based on Sequential Pattern Mining in Cloud Environment", *In Proceedings of American Journal of Engineering and Technology Research*, Volume 11, No.9, 2011.

[5]L.Heung Ki, A. Baik Song and K. Eun Jung "Adaptive Prefetching Scheme Using Web Log Mining in Cluster based Web Systerms". *In Proceedings of IEEE International Conference on Web Services*, 2009.

[6] L.Lin, L.Xuemin, J.Hong, Z.Yifeng, " *AMP: An Affinity-based Metadata Prefetching Scheme in Large-Scale Distributed Storage Systems*", *Technical Report*, Novermber, 2007.

[7] Y. Tin Tin, N. Thinn Thu, "*PC-Cluster based Storage System Architecture for Cloud Storage*", International Journal on Cloud Computing: Services and Architecture (IJCCSA),Vol.1, No.3, November, 2011.

[8]Y.Un-Keun, K.Han-Joon and C.Jae-Young, "Intelligent Data Prefetching for Hybrid Flash-Disk Storage Using Sequential Pattern Mining Technique", *In Proceedings of IEEE/ ACIS Ninth International on Computer and Information Science*, 2006