Graph based model for Object Access Analysis at OSD Client

Riyaz Shiraguppi, Rajat Moona
{riyaz,moona}@iitk.ac.in
Indian Institute of Technology, Kanpur

Abstract

In client-server based storage systems, the knowledge of access behavior of client results in better quality of service. One of its uses is in prefetching to reduce the response time. In this paper, we propose a graph based model for access analysis of objects in the Object based device (OSD) system [1,2]. OSD technology promises to satisfy requirements of enterprise's rapidly growing storage requirements. OSD provides abstraction at the object level, granting object access only after authentication of security tokens which are issued by the OSD manager. We look particularly for prefetching of the object security tokens by an OSD client and perform analysis based on this model. Graph based techniques are commonly used to solve many real world problems [5]-[9]. We present a graph-based client-initiated mechanism for the analysis of object accesses to achieve effective prefetching of objects. We have also devised an access algorithm that works on these graphs and generates an ordered set of objects which are suitable candidates for prefetching along with a given object. We also discuss design and implementation of an OSD client, and its integration with the graph-based model.

1 Introduction

Prefetching is useful for improving response time in client-server architectures. Prefetching techniques bring data a priori at the client side based on data access analysis.

In the simplest approach, aggressive prefetching assumes spatial locality and brings all data near to the data currently being accessed. However, this may lead wastage of network traffic as it ignores user access behavior. In this paper, we consider a selective prefetching mechanism that chooses suitable candidates for prefetching using prediction. Prediction based techniques are used to guess what data is required in the near future based on analysis of data accessed in the past.

Depending upon the location of implementation, prefetching can be server-initiated, client-initiated or proxy-initiated. In server-initiated prefetching, the server keeps a log of accesses by the client. For a given request, future data is predicted by studying accesses made in past. In a proxy-initiated case, a proxy forwards client requests to the server. Prefetching techniques are found to be effective when client provides hints regarding its accesses. In client-initiated case, the client analyzes its own accesses and requests prefetch data.

The object based device (OSD) system [1,2], as a storage area network (SAN) [3] technology, combines the advantages of high-speed, direct-access SANs, and the data sharing and secure capabilities of network attached storage (NAS) [4]. The storage component of the file system which is traditionally implemented by the host software, is implemented by the device itself. Unlike traditional block addressing, this device exports an object interface. Storage management of objects is done internally by the OSD for efficient access. Access to the object is granted after authentication of security tokens for objects. An OSD manager issues security tokens. On receiving request for a security token for an object, it returns a tuple containing the security token and its integrity value. This tuple is termed as the Credentials for the object. The security features in the OSD system introduce the additional overhead of acquiring the required credentials before accessing an object.

In this paper, we propose graph based techniques for selective prefetching of credentials of objects to minimize this overhead. In particular, we consider client-initiated prefetching where the OSD client maintains a log of accesses. Access analysis is carried out on this log to identify frequently accessed objects and ordered sequences of objects. An ordered sequence of objects is also termed as a pattern. These patterns are used for prefetching the objects credentials by the client.

We discuss client side components of the OSD system and integration of graph based model for prefetching of object credentials from the OSD manager. This model can also be applied for selective prefetching of objects from the OSD device.

Graph based techniques are used by several researchers for various applications. James Griffioen and Randy Appleton discussed graph techniques for prefetching to reduce file system latency [5]. In their approach, they create a graph with nodes as files in the file system and edges to represent the access sequence between nodes. Nexus [6] is a weighted graph based prefetching algorithm for metadata servers in petabyte-scale storage systems. This is designed for metadata servers. In this approach, relationship graphs are constructed dynamically by defining successor relationships. George Pallis, Athena Vakali and Jaroslav Pokorny discussed a clustering-based prefetching scheme [7] on a web cache environment. This is a proxy-initiated prefetching scheme where a graph-based clustering algorithm identifies clusters of correlated web pages based on the user's access patterns. Carl Tait and Dan Duchamp discussed detection and exploitation of file working sets [8] for effective prefetching using graph techniques.

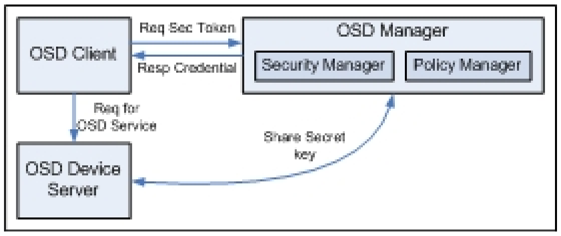


Figure 1: OSD System components

The remainder of the paper is organized as follows. In section 2, we provide an overview of the OSD system. In section 3, we introduce our prefetching model. In section 4, we present the design of an OSD client integrated with our prefetching model. We discuss our implementation in section 6. We conclude in section 7.

2 Overview of OSD system

An OSD system (figure 1) typically consists of the following components.

OSD Client

The OSD client runs applications which operate on the data from the OSD target. It acquires security tokens from the OSD manager and sends request to the OSD target integrated with the corresponding tokens. The OSD client can be high-end processing machine or a NAS head in the real world scenarios.

OSD Manager

The OSD manager is responsible for management of policy decisions for objects in an OSD. It issues security tokens to the OSD client which are encrypted with secret keys shared with the OSD target.

OSD Target

The OSD target is responsible for processing OSD commands after appropriate authentication of the security tokens.

3 Access Graph Model

In this section, we discuss our access graph model for the object pattern analysis. The access graph model uses weighted graphs to create an ordered set of objects. This set is used for the selective prefetch.

3.1 Observation Window

The time duration for which accesses of the objects in the system are observed is termed as the "Observation Window". This factor is dependent on the frequency of objects accesses and changes in the access patterns. Access analysis is carried out at the end of the observation period as a background process.

3.2 Log Information

A log of object accesses is maintained per Observation Window. Past log files are stored for a limited number of Observation Windows. The log contains information specifying each accessed object, its access time, the type of access made on that object, etc. In particular, we consider the log for the prefetch of object credentials. In our case, information about type of access made on the object may not be useful but it might be useful for prefetching of the object data from the OSD target.

3.3 Object Access Count and Inter-reference Interval

The object access count and inter-reference intervals are useful in determining frequency of object access in the Observation Window.

We use this information to prioritize among objects. In our model, we define four priority levels for objects - Very High, High, Medium, Low. Depending upon the size of the Observation Window, a range (in terms of number of accesses) for each priority level is defined. The priority of new object that is not part of any Observation Window is considered as Medium.

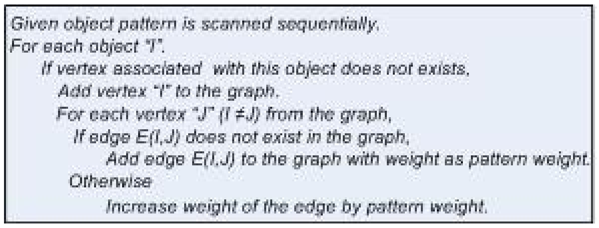
Prioritization is useful for selective prefetching. Objects with higher priority can be brought a priori. Prioritization can also used in the replacement policy. Objects with low priority are chosen as candidates suitable for replacement.

3.4 Object Access graphs

We build an object access graph by identifying pattern of objects accesses. Search for an access pattern, i.e. repeated sequence of objects, is done in the log.

If access patterns do not overlap then the task is simple. For a particular access on a object, all other objects from the pattern accessed after the given object are returned as candidates for prefetching.

It is possible that two or more access patterns overlap. In this case, selection of a pattern can be done by deter-



```
Given object pattern is scanned sequentially.
For each object "I".
    If vertex associated  with this object does not exists,
      Add vertex "I" to the graph.
      For each vertex "J" (I ≠J) from the graph,
        If edge E(I,J) does not exist in the graph,
            Add edge E(I,J) to the graph with weight as pattern weight.
       Otherwise
            Increase weight of the edge by pattern weight.
```

Figure 2: Algorithm for constructing access graph

mining which stored pattern is closest to the current access behavior. This solution is expensive and may cause wastage of prefetch data when accesses of the objects are not according to the determined pattern. Alternatively, objects from all patterns can be selected. However, there is limit on the amount of data that can be prefetched. This implies a need to selectively prefetch objects from all patterns of which the given object is a member.

We propose a graph based solution for selective prefetching. This works by maintaining a weighted directed graph for each access pattern. There is a weight associated with each pattern which is a factor of the number of times the given access pattern is repeated. The factor value is decided depending upon the number of patterns and repetitions of the patterns in the Observation Window.

3.4.1 Construction of Access graphs

We construct a graph for each pattern with a vertex for each object in the graph.

The graph is constructed using method in figure 2.

Figure 3 shows access graphs for various patterns with pattern weight as 10% of repetition count.

It is also possible for the application to give feedback regarding its working set. Objects in the working set are considered as a pattern and the highest pattern weight value is assigned to this pattern to ensure that this pattern has the highest priority.

Construction of directed graphs is useful for defining

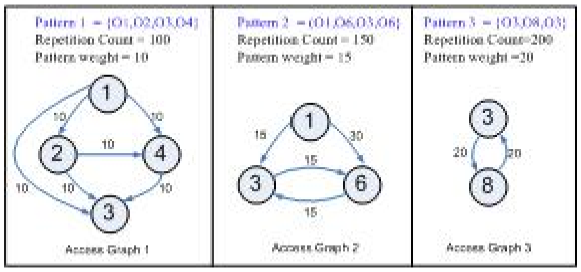


Figure 3: Access Graphs

successor relations among objects. The list of objects obtained by the traversal of directed access graphs gives the set of objects having a high probability of access after the access of the given object.

The access graphs can also be constructed as undirected graphs. In this case, the weight of a resultant undirected edge will be the sum of weights of the directed edges between two vertices. An undirected graph is useful for defining an object working set, i.e. a collection of objects with high probability of getting accessed together.

3.4.2 Merging of graphs

Two access graphs can be merged if their associated patterns contain common objects. The resultant graph contains the union of vertices from both graphs.

The weight for edges of the new graph is assigned as follows. If the edge is common to both graphs, the weight of the edge is the sum of the weights of the given edge from each graph. Otherwise, it is equal to the weight of the edge in the graph where it is present.

Figure 4 shows merging of graphs shown in figure 3.

3.5 Management of Access Graphs

Access graphs are stored in the database at the client side. For each access graph, there is an entry for every edge specifying its two vertices and the weight of the edge.

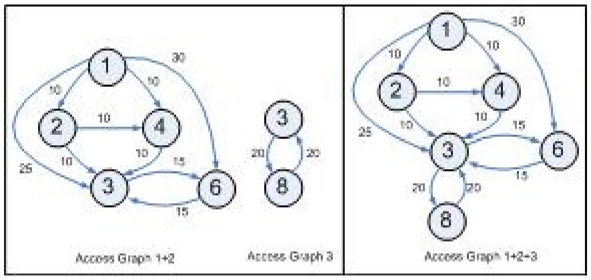


Figure 4: Merging of Access Graphs

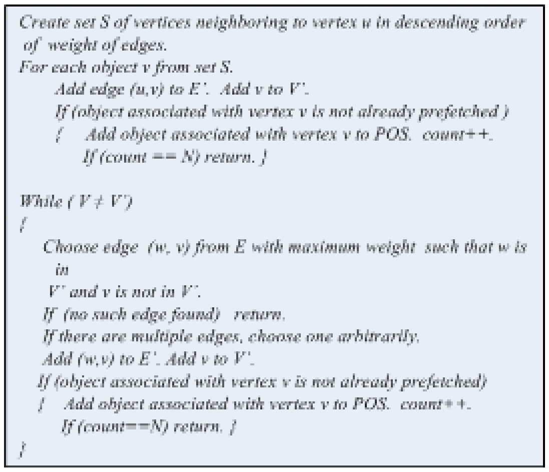
There is also a unique identifier associated with each access graph.

There is another database maintained which stores the age value for each access graph. The age value is increased if the access graph is repeated in the current Observation Window. Age values are assigned in such a way that the values for the access graphs found in the recent Observation Windows are more than those of access graphs found in the past.

Age values are useful for determining replacement policy. It is not possible to store all access graphs from all past Observation Windows. Access graphs with lower age values are chosen as candidates suitable for replacement.

Permanent Access Graph

We call an access graph as a permanent access graph if it is continuously found in the past Observation Windows. Age values are useful in deciding permanent access graphs. Permanent access graphs have very high age values. The use of permanent access graphs is discussed in section 3.6.3.



```
Create set S of vertices neighboring to vertex u in descending order
 of  weight of edges.
For each object v from set S.
      Add edge (u,v) to E'.  Add v to V'.
      If (object associated with vertex v is not already prefetched )
      {     Add object associated with vertex v to POS.  count++.
            If (count == N) return. }

While ( V ≠ V')
{
    Choose edge  (w, v) from E with maximum weight  such that w is
      in
     V' and v is not in V'.
    If  (no such edge found)   return.
    If there are multiple edges, choose one arbitrarily.
    Add (w,v) to E'. Add v to V'.
   If (object associated with vertex v is not already prefetched)
   {    Add object associated with vertex v to POS.  count++.
        If (count==N) return. }
}
```

Figure 5: POS construction Algorithm

3.6 Selective Prefetch using Access graphs

3.6.1 Construction of Prefetch Object Set

As mentioned earlier, the size of data that can be prefetched is usually limited. Selective prefetching is done on the basis of weights of edges. An ordered Prefetch Object Set (POS) with objects in descending order of prefetch preference is constructed using an access graph. Depending upon the size of the prefetch buffer, prefetch data for the objects from this set is requested.

When an object is accessed, the access graph containing the given object is searched. If it is not found, prefetching is not done.

The POS is constructed using the algorithm shown in figure 5. This is a modified form of Prim's minimum spanning tree algorithm [10]. Initially, all immediate neighbors of the given object are considered. Subsequently, a modified Prim's algorithm is used where the next vertex is chosen having the edge with the maximum weight.

For example, consider the generation of the POS for vertex 2 of the graph constructed in step 2 of figure 4. Initially, immediate neighbors vertices 3 and 4 are chosen. This creates a partial tree containing vertices 2, 3, 4. The modified Prim's algorithm is applied on this partial tree which chooses vertex 8 (as edge 3-8 is of maximum weight) and then vertex 6. Thus, the POS generated for vertex 2 is {3, 4, 8, 6}.

It may be seen that the edges corresponding to the objects with repeated occurrence in a pattern have higher weight. This gives higher preference for prefetching such objects.

3.6.2 Permanent POS

It is possible to save time for building a POS for frequently accessed objects by constructing it a priori and storing it in the database. Such a POS is known as a Permanent POS.

3.6.3 Participation of OSD Manager

It is also possible for the OSD manager to participate in the process of prefetching of credentials. The OSD client can send permanent access graphs to the OSD manager. When an object from the permanent access graph is accessed, the client does not need to build the POS. This task is done by the OSD manager. The OSD client can also send the permanent POS. This is helpful in reducing the workload at the client side.

This is also useful when some OSD client is not smart enough to generate and send its POS. The OSD manager can merge permanent access graphs sent by the other smart clients and use them to generate POS for this client.

4 OSD Client Design

In this section, we describe the working of OSD client integrated with Access Graph Model. We will look at the design of modules required for selective prefetch of object credentials from the OSD manager.

An OSD client consists of various modules organized in Application layer, Object layer and SCSI layer, as shown in figure 6.

4.1 Application Layer

Application layer consists of programs and utilities that operate on OSD data.

4.1.1 OSD Application

OSD application is an OSD object aware application which manipulates its data in terms of objects. Instead of using traditional read/write system calls, it calls object read/write functions of object layer to operate on the OSD data. This can be a file system aware of the OSD system (OSDFS) or a utility program for manipulation of objects. Intelligent OSD application can also provide feedback regarding the object working set to object layer (section 4.2).

4.1.2 Traditional Application

A traditional application is based on the block architecture. It is unaware of OSD system and objects, and operates on data using read/write system calls. An object mapper is required which maps block based I/O to object based I/O in order to port such applications to the OSD system.

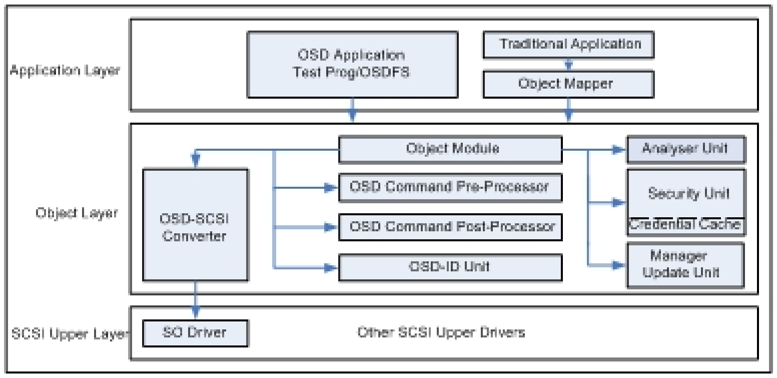


Figure 6: Architecture of an OSD Client

4.1.3 Object Mapper

An object mapper allows block based application to operate on OSD device. It is responsible for converting system read/write calls to object read/write commands in the object layer. This module especially maps files/blocks to the objects. Object mapping can be either at the block level where data blocks are mapped to the objects, or at file level where files are mapped to the objects. Object mapper can also provide feedback regarding object working set to the object layer.

4.2 Object Layer

OSD commands are submitted to the SCSI stack as SCSI commands. Object read/write calls made by the application layer need to be mapped to SCSI calls. This mapping is done in the object layer. Along with this mapping, an OSD request needs additional processing. A similar processing also needs to be done for the command response before passing it to the application layer. The object layer includes various modules such as the following.

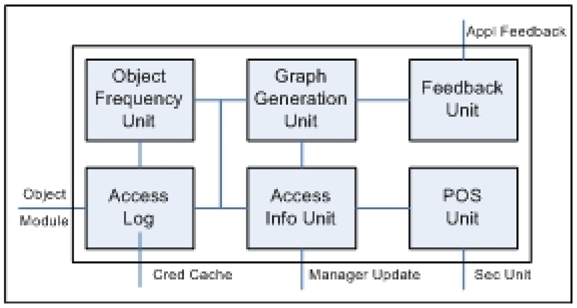


Figure 7: Analyzer Module

4.2.1 Object Module

This module exports object read/write functions which are called by the application layer modules. This module acts as the communicator between upper layer and object layer, and is responsible for sequencing of operations in the object layer by invoking various modules.

The Object Module also sends information about object access to the Analyzer module. Access information specifies object information, type of access and name of the application accessing this object.

4.2.2 Analyzer Module

Figure 7 shows various units of Analyzer Module.

Access Log

Information about current object access sent by Object Module is added to the Access Log. The Access Log is maintained in a very simple manner since it is updated for each access of every object. We propose to implement it as simple text file for each Observation Window with each line specifying time of access, information about application accessing it, object ID information and type of access.

Object Frequency Unit

This module is responsible for prioritization among objects. It works on the log file associated with the current Observation Window. It assigns priorities to each object depending on the number of accesses and average interval time. Priority information is stored in the database maintained by Access Information Unit.

Object frequency unit is activated once every Observation Window, at the end of observation period.

Graph Generator Unit

Graph Generator unit is responsible for construction of access graphs. It works on the log file associated with the current Observation Window and searches for the access patterns. As there is a high probability of getting a pattern around frequently accessed objects, it takes feedback from Object Frequency unit regarding accesses of high priority objects.

The graph generator unit constructs a directed graph for each access pattern which is called as access graph for that pattern using method described in section 3.4.1. A Pattern Weight is also assigned to each access graph.

After construction of access graphs, these graphs are merged if their associated access patterns contain common objects. Final access graphs are stored in the database at Access Information Unit.

Graph Generator Unit is run once every Observation Window at the end of the observation period.

Feedback Unit

Application layer can also help in the process of pattern analysis by providing information regarding the object working set. Feedback unit exports object calls that allow application to provide object working set. It also submits each working set as a pattern with very high repetition count to the Graph Generator Unit.

Access Information Unit

Access Information unit maintains databases for Object Priority, Weighted Access graphs, Age of Access graphs and Permanent POS. These databases are updated once every Observation Window at the end of the observation period.

The "Object Priority" database contains an entry for each object found in the Observation Window with its priority. Entries are added by Object Frequency unit during analysis of Access Log. Credential Cache (section 4.2.7) queries priority information of objects while adding them to the Credential Cache. Security unit (section 4.2.6) queries this database before sending prefetch requests for high priority objects to the OSD manager. POS unit also queries this database for building permanent POS associated with the high priority objects.

The "Weighted Access graphs" database stores access graphs. A unique identifier is assigned to each access graph. Access graph information is maintained in the form of edge list representation with a weight assigned to each edge. Entries are added by the Graph Generator unit at the end of observation period. This database is queried by POS unit for construction of POS for given object.

The “Age of Access graphs” database stores information about each access graph (with unique identifier) and its age. Age is computed using method described in section 3.5. Graph Generator unit adds a new entry for every new access graph found in current Observation Window. It also updates age values for old access graphs. This database is queried by POS unit for construction of permanent POS. This is also queried by Manager Update unit to update OSD manager with permanent access graphs.

The “Permanent POS” database contains POS for high priority objects belonging to permanent access graphs. POS unit adds entries in this database at the end of the observation period. Manager Update unit also queries this database to update OSD manager with permanent access graph.

POS Unit

POS unit is responsible for generation of prefetch object set. Request for selective POS of an object is sent to this unit by the Security Unit (section 4.2.6). First, this unit queries the “Permanent POS” database to check whether POS for the given object exists. If it doesn’t exists, it queries the “Weighted Access graphs” database for access graph containing this object. POS is generated using algorithm described in section 3.6.1.

4.2.3 OSD ID Unit

In our implementation, OSDs are implemented as SCSI devices. We have used SCSI device id to identify the OSD. OSD ID unit is responsible for management of SCSI device id for OSD devices in the system.

4.2.4 Object Command Pre-processor

The OSD requests from the application layer are preprocessed before they move on to the SCSI layer. Preprocessing includes the following.

- Invoking the security unit for obtaining security token.
- Placing the request for required metadata information of the object to the OSD device. This information may be modified as a side effect of command execution, for example to include the access time. Such modifications are updated to the OSD manager.
- Generating MIC for OSD request and attaching it with OSD request.

4.2.5 Object Command Post-processor

This module is responsible for processing metadata information requested by the pre-processing unit. Post processing is carried out after command execution and includes updating the Credential Cache and the security manager with modification in the metadata information as a side effect of the command execution.

4.2.6 Security Unit

Security unit is responsible for implementation of security features at the client side which includes getting credentials for the object. For an OSD request to the given OSD device, SCSI device ID is obtained from the OSD ID unit. Credentials are first searched in the Credential Cache. If credentials are found but permission rights do not allow performing a given operation, requested OSD command is rejected with an error. If credentials are not found, POS for the given object is obtained from the Analyzer module. In this case, credential request for the given object along with objects in POS is made to the OSD manager. Credential request contains OSD-SCSI ID, object information, user information and POS set.

Credential response contains credentials for the given object and the objects from POS set. Entries for all objects are added to the Credential Cache as a background process.

4.2.7 Credential Cache

Credential Cache is a local store used to store recently accessed credentials. Entries are added to Credential Cache from the response of the security manager. Entries are removed after the execution of remove commands or when credentials expire.

An entry in the Credential Cache consists of object information (OSD-SCSI ID, object type, part-id, object-id), user information (uid, gid), credentials and priority of object. Priority information of objects is obtained from Analyzer module. If the priority information is not found, given object is assigned medium priority. We use hash queues for management of the Credential Cache entries. Priority based replacement policy is used.

4.2.8 OSD-SCSI Module

This module is responsible for encapsulation and decapsulation of OSD request into SCSI control descriptor block (CDB).

4.2.9 Manager Update Unit

OSD manager maintains database containing access control information and metadata for objects. This database need to be updated after execution of CREATE/REMOVE commands at the OSD client. This task is done by the Manager Update unit. A request from the OSD client to the OSD manager is sent by the Manager Update unit for updating database at OSD manager side as a side effect of command execution. Beside this, the Manager Update unit also sends information about permanent access graphs and permanent POS.

4.3 SCSI Upper Layer

SCSI upper layer is implemented as a part of the Linux SCSI stack [11]. It implements SCSI upper layer driver for the OSD through SO driver. Like other SCSI upper layer drivers such as those for SCSI disk (sd) and SCSI tape (st) devices, it is responsible for detecting and managing OSD devices in the host system. OSD command is submitted to this layer module. OSD ID unit queries the SO driver using ioctl system call to get list of all OSD devices.

5 Our implementation

Our current implementation includes manager and client side components of the OSD system. We are using IBM OSD initiator project [12] which consists of implementation of SO module and OSD-SCSI module. The OSD manager and client side components are built on the top of it. At the OSD target side, we have our own open source implementation of T10 compliant OSD simulator [13].

6 Conclusion

In this paper, we discussed use of graph based techniques for prediction of future accesses from analysis of past access history. We built access graphs using the log of object accesses. Our algorithm works on these graphs and generates set for prefetch data.

Features of our graph model can be summarized as the following.

- Construction of weighted directed graph for each pattern allows to define successor relationship between various objects within the graph. Weight of each edge describes strength of this relation.

- Merging of access graphs allows to define strength of this relation across multiple patterns. Unlike other approaches [6,8] where prediction is done considering single access pattern, merged access graphs allows to consider multiple patterns for prediction. This can make prediction more accurate.
- Our access algorithm to generate ordered prefetch object set is a modified form of Prim's minimum spanning tree algorithm. Initially, prefetch object set is constructed using all immediate neighbors of the object giving them high priority. Later on using a modified Prim's algorithm for maximum weight allows adding of objects having high priority of access after accessing a object from partial prefetch set.

We have described various subunits in the OSD client system and discussed about its integration with this model. In particular, we have discussed its use in prefetching of the credentials from OSD manager.

References

[1] Eric Riedel. OSD Architecture and Systems. SNIA Technical Tutorial. April 2006. http://www.snia.org/education/tutorials/2006/spring/storage/Object-based_Storage_Devices-OSD_Architecture_and_Systems.pdf.

[2] T10 Technical Committee. Information technology-SCSI Object-Based Storage Device Commands-2 (OSD-2). Working draft, project T10/1729-D, Revision 2., July 2007. http://www.t10.org/ftp/t10/drafts/osd2/osd2r02.pdf.

[3] Heng Liao. Storage Area Network Architectures, Technology White Paper. PMC-2022178. Issue 1: April, 2003. http://www.pmc-sierra.com/cgi-bin/document.pl?docnum=2022178.

[4] Garth A. Gibson and Rodney Van Meter. Network Attached Storage Architecture. Communications of the ACM, Nov. 2000, vol. 43, issue 11. ACM Press New York, NY, USA. pp. 37-45.

[5] James Griffioen, KY Randy Appleton. Reducing file system latency using a predictive approach. Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1. Boston, Massachusetts. pp. 13 - 13

[6] Peng Gu, et. al., Nexus: A Novel Weighted-Graph-Based Prefetching Algorithm for Metadata Servers in Petabyte-Scale Storage Systems. Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06) - Volume 00. Singapore. 16-19 May 2006. pp. 409 - 416.

[7] Pallis G., et. al., A Clustering-based Prefetching Scheme on a Web Cache Environment. International Journal Computers & electrical engineering, Elsevier, 2007.

[8] Carl D. Tait et al., "Detection and Exploitation of File Working Sets," International Conference on Distributed Computing Systems, IEEE, Arlington, Texas, USA. May 20-24, 1991. pp. 2-9.

[9] Murali Annavaram, Jignesh M. Patel, Edward S. Davidson. Call graph prefetching for database applications. ACM Transactions on Computer Systems (TOCS) archive Volume 21 , Issue 4, (November 2003), pp. 412 - 444.

[10] R.C. Prim. Shortest connection networks and some generalizations. Bell System Technical Journal 36 (November 1957), pp 1389-1401.

[11] Douglas Gilbert. The Linux 2.4 SCSI subsystem HOWTO.
http://sg.torque.net/scsi/SCSI-2.4-HOWTO/.

[12] IBM Inc., IBM OSD Initiator.
https://sourceforge.net/projects/osd-initiator.

[13] Riyaz Shiraguppi, Implementation of IITK OSD Simulator.
https://sourceforge.net/projects/iik-osd-sim