修士論文

Balanced Placement on a Cluster File System using Request Graph Early Reservation

ファイル関連グラフを用いたクラスタファイルシステムの負荷分散に関する研究

指導教官
松尾 啓志 教授
津邑 公暁 准教授

名古屋工業大学大学院工学研究科
修士課程情報工学専攻
平成 19 年度入学　19417643 番

Honsali Khalil

平成 20 年 2 月 5 日

Abstract

In this work, we investigate a novel method for balanced placement for a set of related files on a cluster file system. The propose method assumes knoweldge of the file set in early reservation settings. The goal of the placement is two fold: first, to balance the workload on the cluster file system's I/O nodes, and second to maintain the relations of the file set in order to minimize the request overhead for both client and server.

Requests for a set of related files are modelled as a request graph with given properties such that vertices correspond to request files' attributes and edges correspond to probabilities of request dependencies between files. This is similar to the graph shapes used in task-graph scheduling algorithms, since a request of a file is similar to a task. Using this graph model, a placement algorithm was developped in order to optimize placement of the graph such that file attributes and their relationships are taken into consideration, since both affect the total workload on the system during request arrival.

The proposed placement is a two step algorithm : the first step is a rating phase that traverses the graph in breath first search and assigns file requests with highest priority to storage nodes, based on requests weights and storage nodes metrics. The second step si a novel advice model that is used to affect the placement decision when two files are related, i.e. a request for a given file has a high probability to request another file. The requesting file advices the requested file by adjusting its rating, in order to have it assigned on the same storage node if the system balance allows it to. Using this approach, the algorithm tries to place the files while satisfying both constraints: file request relationships and cluster balance.

A cluster simulator was developped to test the behavior of the presented algorithm. Randomly generated request graphs with various properties are placed on the cluster using the present work's algorithm and compared to classical RoundRobin and Weighted Scheduler algorithms. Client requests that stresses the structure of the request graph were simulated. The resulting cluster behavior was measured for the varying parameters and the result have shown that our approach performs usually better and at worst similar to classical algorithms in terms of cluster balance, and outperforms them all in terms of graph balance.

Acknowledgements

There is no God but God, to God only we pray, and there is no way or power without God.

Reverence to my parents, may God reward them with mercy and more++

Respect and Gratitude to my master, Matsuo-sensei, for his valuable teachings.

Peace and Thanks to the Japanese Ministery of Education for their financial support.

Also, thanks to the International Division of NIT for their help.

Peace to all friends, especially Hakeem.

Peace to all teachers and students.

To start, I refer to [Al Fati7a], recited at least 17 times in 5 daily prayers.

Balanced Placement on a Cluster File System using Request Graph Early Reservation

目次

表一覧

1 Introduction

1.1 Objective

Over the past decade computer clusters have emerged as the dominant architecture in high performance computing, combining the power of hundreds or thousands of commodity off-the-shelf computers that can scale to Peta-bytes of storage. Cluster File Systems are one of the core software components on such architectures and many solutions are already available for different workload environment, providing high throughput, failure tolerance and incremental scalability. Moreover, with internet data boom, cluster file systems are being used in web-based service environments. In such scenarios, data is most often stored in large data structures with complex relationships and this implies growing demand for higher lever file system designs that understand the nature of data being stored not only to support efficient processing but also to guarantee a balanced system and hence a better quality of service.

Traditionally, the utilized approach for the server to be aware of data patterns was to perform static and/or dynamic statistical analysis of client requests on one side and of I/O nodes status on the other side, in order balance the system. But this approach suffers from latencies between the measured workload and the actual workload, due to the time lag between request arrival, processing and real-time system status. Moreover, with rigid semantics the server is not aware of the relationships of the data being stored. Therefore to intelligently manage workload a change in the client/server interaction is needed.

We propose a novel approach for balancing load on a cluster using information provided by the client as early request reservations for a set of files. Since the file system client is the closest software entity to the user, it is possible to assume that the client knows best the user behavior and hence associated datasets structure, for instance by the means of statistical accounting of user file operations and usage patterns. Such information as file size, directory structure, read/write operation frequency, file expiry rate, link distribution and frequency ...etc, can be acquired by the client before data is uploaded to the server. This information could be very useful to the server if delivered beforehand and could greatly impact the cluster overall system performance.

The present thesis statement is: Assuming the Server can receive early information

about the datasets to be stored, can it manage more efficiently its available resources while maintaing datasets structures?

1.2 Overview

The structure of this report is as follows:

In chapter 2, a background is presented in order to introduce the context of the current work and explain the problem targetted and the technicalities on which the solution is based. First, file systems and computer clusters are introduced. Then, the file system designs that are specific to computer clusters are explained, including operational principles, key design issues and concepts, popular cluster file system software and related technologies. Finally, a brief literature overview of the theories on load balancing and resource placement are also presented, including related work of task graph scheduling and data partitioning.

In chapter 3, the models and computations underlying at the core of the present work are explained. The targetted cluster model is defined. The request graph model, its properties and important equations are defined discussed. Different senarios for placement of the request graph are examined, including the target scenario. The proposed approach, computation models and algorithms for balanced placement of the request graph are defined and discussed.

In chapter 4, the implementation details of the request graph, the cluster simulator and the request graph generator are given with supporting pseudo-code.

In chapter 5, the evaluation strategy with details about the experiemntal apparatus and parameters are described. Results of experimentation for the cluster balance and graph partioning are presented and discussed, including comparison of the proposed solution with three other algorithms.

In chapter 6, conclusion with discussion about the current research and future directions are provided.

Finally, the bibliography and appendix sections.

2 Background

2.1 File Systems

A file system, as the name implies, is a software system, i.e., a set of programs that manage files stored on a hardware medium. File systems are everywhere because they are necessary to every software working with files, hence it is an extra large area of research as old as computer science.

A comprehensive introduction to the key terms and concepts associated with the subject is provided in [1], of which the most important are given below:

- **Disk**: The storage medium, the hardware, on which files are stored. A disk is the primary bottleneck of speed on a computer lagging behind several orders of magnitude from the CPU, the RAM and the network respectively. Continuously new technologies aim at reducing the speed gap, the latest of which is the Solid State Drive (SSD). A disk block or sector is the minimum hardware storage unit and is mostly around 512 to 1024 bytes in modern hardware.
- **Block**: The smallest unit writable by a disk or file system. Everything a file system does is composed of operations done on blocks. A file system block is always the same size as or larger (in integer multiples) than the disk block size, however the size depends on the scale considered, when 512 to 1024 bytes are usual block sizes for local usage, high performance systems may use up to 64 Megbyte block sizes. The reason for smaller block size is that the majority of the Operating System and text files stored on disk is small, the reason for larger block size is the large datasets associated with the context.
- **Extent**: An range of blocks, contains a starting block number and a length of successive blocks following it on disk, also known as block runs.
- **Attribute**: A name (as a text string) and value associated with the name. The value may have a defined type (string, integer, etc.), or it may just be arbitrary data.
- **Metadata**: A general term that refers to information about data but that is not part of it. Metadata in filesystems is information about the files, such as the length, number of blocks, Stores file owner, POSIX file permissions, Creation timestamps, Last access/ read timestamps, Last modification of content, This

copy created, Last metadata change timestamps, Last archive timestamps, Access control lists, Security/ MAC labels, Extended attributes/ Alternate data streams/ forks, Checksum/ ECC ...etc. However, not all systems do support all of the above..

- **Posix Semantics** : POSIX stands for "Portable Operating System Interface", as the name suggests it is a family of standards specified by the IEEE organization with the goal of having compatible interfaces between Operating Systems. The designation number is 1003. POSIX covers a wide range of concepts, guarantees for thread concurrency, high performance, but most importantly I/O operations [2].

The next question that comes into mind is how a file system stores the data it contains? This depends on the type of file systems. Let us review the two most popular systems:

NTFS: is the *defacto* file system for windows operating systems and the successor to FAT. Beside performance and security related improvements, NTFS has an advanced metadata support. The Master File Table (MFT) contains all the metadata of all of the files, and is stored as B-Tree like strucutre for fast indexing.

Ext3: or the third extended file system, is linux's default file system and an improved version to ext2, a journaled file system. Its main feature is reliability and scalability. It belongs to the family of unix file system that use the concept of **I-node**. The I-node, or File Control Block (FCB), is an efficient data structure from the B-Tree family that stores all the necessary metadata about a file, in addition to the pointers to the location of the file.

2.2 Computer Clusters

A computer cluster can be definded [3] as "a set of computers that are connected via a Local Area Network (LAN) and that appear to the external world as a single entity". Usually the computers are independent commercial-off-the-shelf (COTS) machines with a single or multiple processors or cores. A Beowulf system is a typical configuration using Personal Computer (PC) machines with an open-source Operating System (OS) installed, typically Unix/Linux. also called PC Cluster, such as the one shown in Figure 1.

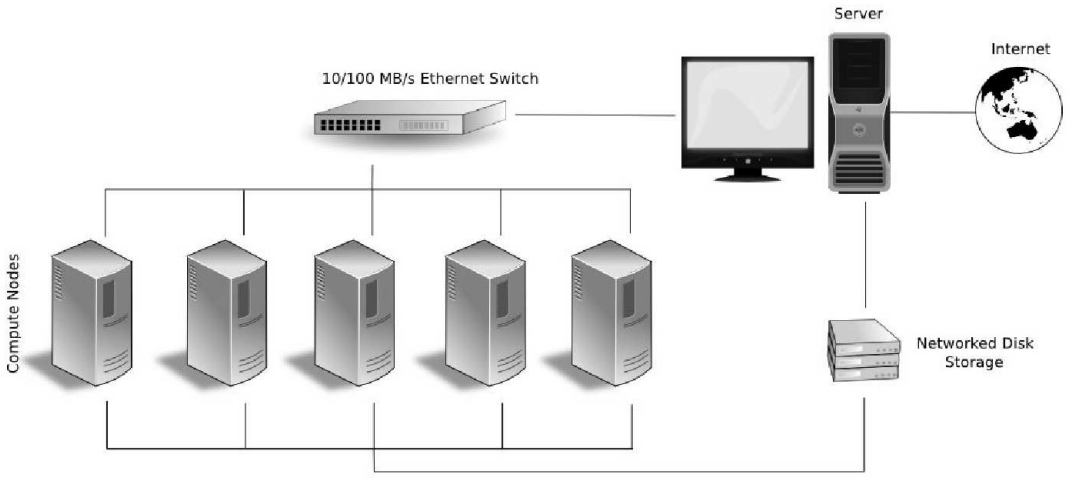

図 1: A typical Beowulf Cluster

A computer Grid is a collection of clusters geographically disparate.
Computer cluster may be tuned to serve specific functions:

- **High performance** clusters, which are also referred to as computational cluster systems. These systems are normally utilized to support very large data volumes (of computational processing). In such an environment, a parallel file system distributes the processing resources across the nodes, thereby allowing each node to access the same set of files concurrently (via concurrent read() and write() requests).
- **High availability (HA)** clusters, which are designed for fault tolerance or redundancy purposes. As these clusters normally use one or more servers (for data processing), the servers in the cluster are able to assume processing responsibilities in case that one or more of them goes down.
- **Load balancing** clusters distribute the workload as evenly as possible across multiple server systems, such as web or application servers, respectively.
- **Database** cluster systems, such as Oracles Real Application Cluster (RAC) platform, which introduce many of the cluster file system features into the application-layer itself.
- **Storage** cluster systems, which are utilized between Storage Area Network (SAN) components and server systems with different operating systems. These systems provide shared access to data blocks on a common storage media.

Q: Why is it that computer clusters are being used outside the realm of scientific computing to replace mainframes at the enterprise ?
A: The advantages of clusters are numerous, starting with the cost-to-performance ration, clusters can provide similar and better performance than mainframes at low-ercost, moreover it suffices to add new computer nodes to scale up, while mainframes can not.

Below are some properties of clusters, taken from the literature:

- **Scalability**[4]: Scalability refers to the system capability to support increasing numbers of users, and hence data. In computer clusters, the intuitive way to scale up the system is simply by adding ressources, either as components (CPU, memory, disk, network) or as whole computer nodes. In this case, it is called 'incremental scalability', and is probably a property that only computer clusters can support. In a company as Amazon, these numbers may give an idea about scalability: 10,000 servers, 1,000,000 clients, 10 million requests / day, 100,000 concurrent active sessions
- **High Availability** [5]: Always-On Experience For the case of the mainframe computer, even if there is infinite money to scale up the system using components, the server is still a single point of failure: if it fails the whole software and data is inacessible. Computer clusters on the other hand, because of their decomposition, cannot fail all over at once. According to Google's datacenter information, at most one third of the machine is inavailable at some given moment. Consequently, a great effort is spent on how to prevent loss of data even if computer nodes are lost, and this is achieved via replication [6]. Clustering is indeed a solution of high availability by providing redundant servers and hence redundant data.
- **Fault Tolerance**[7] Highly available data is not necessarily strictly correct data hen a server instance fails, the service is still available, because new requests can be handled by other redundant server instances in the cluster. But the requests which are in processing in the failed server when the server is failing may not get the correct data, whereas a fault tolerant service always guarantees strictly correct behavior despite a certain number of faults. Failover is another key technology behind clustering to achieve fault tolerance. By choosing another node in the cluster, the process will continue when the original node fails. Failing over to

another node can be coded explicitly or performed automatically by the underlying platform which transparently reroutes communication to another serve

- **Load balancing** Load balancing is one of the key technologies behind clustering, which is a way to obtain high availability and better performance by dispatching incoming requests to different servers. A load balancer can be anything from a simple Servlet or Plug-in (a Linux box using ipchains to do the work, for example), to expensive hardware with an SSL accelerator embedded in it. In addition to dispatching requests, a load balancer should perform some other important tasks such as session stickiness to have a user session live entirely on one server and health check (or heartbeat) to prevent dispatching requests to a failing server. Sometimes the load balancer will participant in the Failover process, which will be mentioned later.

When a computer cluster is beyond a certain scale and performance, it becomes a supercomputer. The organization top500 []

2.3 File Systems x Computer Clusters

This is the combination of file systems on a network of computers, which is a logical evolution to DAS (Direct Attached Storage), which is limited in terms of capacity and difficult in terms of price.

2.3.1 Operational Overview

File Systems on Computer Clusters span a range of functional objectives and consequently there are different design approaches for each functional type. However, many share the basic operational principles shown in Figure 2.

At first the client sends a request to the metadata server asking for the location of a file in order to access it. The metadata server is the central authority that manages the cluster I/O nodes, keeping track of their status and assiging files to them. It maintains a metadata table that primarily is a mapping of files, metadata, blocks, replicas and the nodes on wich they are stored. The server looks up the requested file metadata on that table as it is stored on a Database or a data structure, and returns a metdata response to inform the client where the file can be accessed. The client can subsequently cache this information. Next, the client can finally issue the file operation and I/O transfer directly with the concerned node. The transfer may

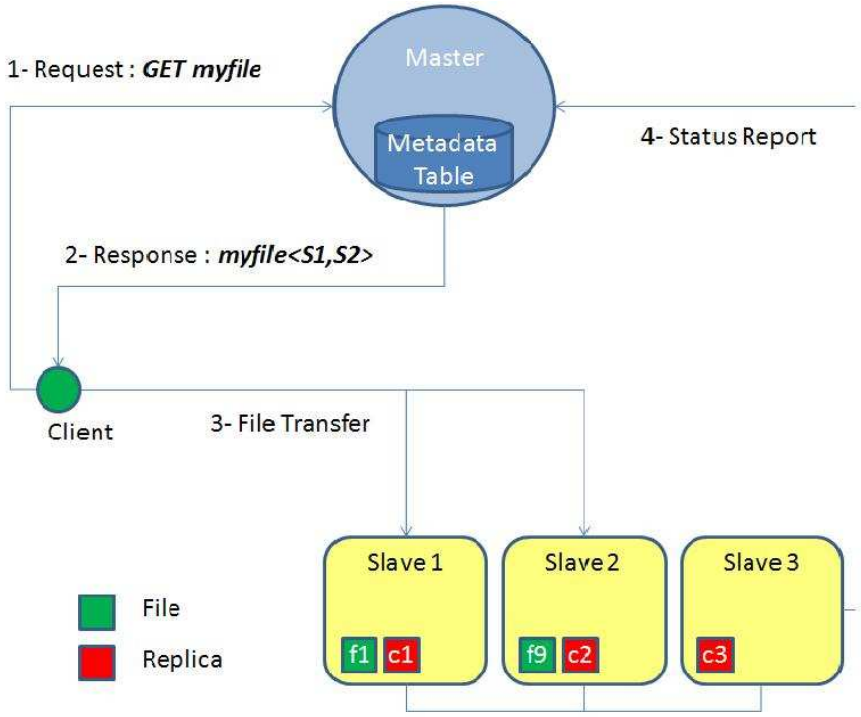

図 2: Operational lifecycle of a Clustered File System

occur in parallel if the file is partitioned into several nodes and usually it is so in order to improve transfer speed. I/O nodes periodically sends a status report (also called heartbeat) to the metadata server to report on ressources and workload usage. This information is crucial to the server for balancing node, taking file placement decisions and to know if a node is available or not. If a node fails, copies of files called replicas are maintained to guarantee data availability for the client.

2.3.2 Taxonomy

Unfortunately, there is no concensus on the naming of file systems for the cluster environment. We base our terminology section based on the literature available.

- **Network Attached Storage (NAS)** is a technology to attach storage devices to a network and mount them via network file servers such as NFS.
- **Storage Area Network (SAN)** is a technology in which storage devices are connected through a network (usually fiber channel) but appear to the OS as locally attached.
- **Shared-Disk File Systems**, also known as SAN File System, is a file system shared by multiple computers (i.e., cluster nodes) and

- **Cluster File Systems**, is a term replacing Shared-Disk File System, meaning the same thing. However, in a Cluster File Systems, nodes may share a local storage.
- **Parallel File Systems** is a flavor of Cluster File Systems in which file transfer between the client and the server occurs simultaneously, hence the word parallel, and are mainly used for high performance environments.
- **Distributed File System** are similar to clustered FS but adopt a different approach, in that clients don't have direct access to the block interface to storage but rather interact with the remote OS via a protocol, which seems to provide better security.

Besides differences in naming issue, some key concepts should be understood:

- **Symmetric vs. Assymmetric**

  In a symmetric FS both the client and the server can reside on the same node. Moreover, a single metadata server is not required anymore, any participating node can share the role of a metada server.
- **Consistency**

  It regroups all the issues regarding the state of system and data being correct even against hardware failures, software crash, multi-threading, caching..etc. From the database field, consistency is measured by the ACID properties (Atomic, Consistency, Isolation, Durability), if clustered file systems, either strong (pessimistic) or weak (optimistic) consistencies are supported, where strong enforces the ACID properties via locking; the weak tends to relax them, and it is the most adopted approach for its simplicity and effectiveness.
- **Stateless vs. Stateful**

  The notion of state is self-explaining: the server is stateful when it keeps track of the information about client requests, including file metadata, handles to on-going operations ..etc, this is a heavy load on the cache operation and mostly difficult to impelment. Stateless on the other hand are simpler to implement, because they don't inforce any consistency issue. Pure decentralized (symmetric) are easier to support stateless.
- **Transparency** If the Client knows server/file mapping, the client contact server directly, hence smaller query processing per client request, but there is risk for

resource contention and/or server failure as the number of client outgrows the server. On the other hand, if the client contacts any server, there is location transparency and better load balancing maintained with an incremental scalability, but the query processing time is longer.

- **Redundancy, Replication**

  It affects reliability. Redundancy is the same as replication, it consists of storing multiple copies of the same file, called replicas, on different cluster nodes, this way even if a node fails the file is not lost. Explicit replication involves failing the file transfer until the replicas are created, while Lazy replication is a delayed form and more relaxed.

- **Striping , Sieving**

  Both affect Throughput, or speed of data transfer. Data Striping is a method used by parallel version of clusterd file systems and consist of sending the file blocks to multiple nodes simultaneously instead of storing the whole file on a single node, which improves the throughput. Data Sieving on the other hand, aims at reducing the request overhead by read contiguous chunks of data from disk even if the requested data is just part of it, this in order to avoid doing multiple random accesses.

2.3.3 Software

Many commercial/closed file systems for the clusters have been available since the 90s and until today, such as : Amazon's S3[8] and Dynamo[9], Google's GFS[10], HP's HFS, SGI's XFS[11], Microsoft's DFS[12] and Panasas [13] among others.
Table 1 lists a family of file systems for the cluster that are not only performant but also open source, which advocates this development approach:

2.4 Placement, Scheduling and Load Balancing

Algorithms developped in the area of Scheduling [16] and Data Placement [17] are still used today for their simplicity and efficiency:

Data Placement

FirstFit: skip the file requests with data size greater than available and assign the first one to fit

LargeFit: sorts data and push them into placement queue by largest first, has more

表1: Types of File Systems for the Cluster

System	Architecture	Notes
AFS/Coda [14] (CMU)	Distributed, Symmetric	POSIX compliant supports offline operations
Ceph [15] (U. California)	Distributed, Parallel	
GFS (RedHat)	Clustered , Assymmetric	almost POSIX
GPFS	Clustered, Parallel	(IBM) used on Blugene supercomputer
Gluster	Clustered, Symmetric Parallel	(Gluster)
Hadoop (Apache), Yahoo!	Distributed, Asymmetric	not-POSIX Clone of GoogleFS
Lustre	Clustered, ASymmetric Object-based	(SUN) used on Blugene supercomputer
PVFS2 (Clemson U.)	Clustered, Symmetric Parallel	POSIX, MPI-IO

complexity than above

SmallFit: same as above but with inversed priority

BestFit: greedy method to search for all the best receiving node

Job Scheduling Algorithms

First Come First Served (FCFS): as the name implies, it schedules jobs on a first-come-first-served basis.

Shortest Job First: gives priority to the jobs with the minimum workload requirements

Multi-Level Queue Priority: push jobs of different sizes into different queues corresonding to each size, providing granularity and control

Some more complex algorithms are developped usign ANT routing, DCOP using agents..etc but they usually share the processing overhead as a drawback. Still, heuris-

tics is another method to compete with or improve classical algorithms.

A straightforward technique that is valid for heterogeneous environments is the periodic system status report sent by participating nodes [18]. The report information metric consists of a combination of CPU cycles and disk capacity to represent workload and capacity status. Report message is either broadcast in symmetric systems, or it is sent to the central authority in assymmetric systems.

Although it may not seem so, randomization in load distribution is still an efficient technique used in distributed systems, specifically in clusters and peer-to-peer [19]. Randomization's most attractive advantage is the lighter processing and stability of behavior over different workload environments. Several techniques are available, noteably Random and Pseudo-Random hashing.

Request routing or redirection is a mean of dynamically distributing load on a cluster . For the case of systems where any server can process the request, web request routing [20] or DNS rotation [21] is applied. For the case of systems where servers are specialized, requests are examined and routed to the specialized server sets [22], an example of specialization can target the file size, so each server group stores specific file sizes. Where file sizes are large, such as in multimedia servers, the technique of striping is applied in order to distribute large I/O requests equally among shared-disks and thus balance the system [23].

Another specialization approach is termed "functional decomposition" and is described in [24].

2.5 Related Work

Task Graph Scheduling[25, 26] is an extensive area of research that grew along with the adoption of clustered systems in scientific applications. The basic goal is to schedule a set of related tasks on a set of available processors in order to minimize total processing cost, known as makespan. Related tasks are modelled as a Directed Acyclic Graph (DAG) in which vertices represent processing tasks, and edges represent dependencies between tasks. The weight of the vertices is the processing cost, in terms of CPU, memory, disk and network resource consumption; and the weight of the edges is the communication cost, in terms of priority and/or communication time. Figure 3 illustrates a simple example of a task-graph scheduling problem.

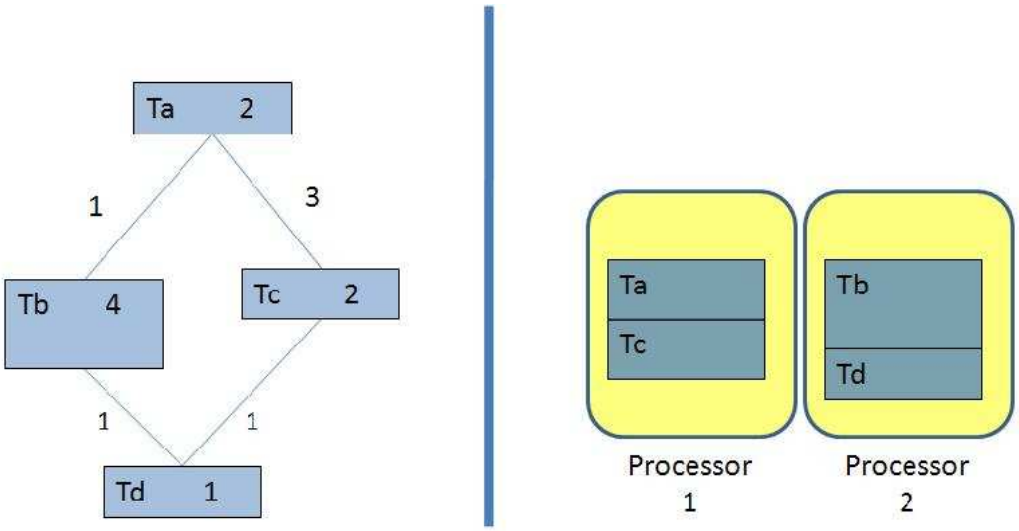

図 3: A simple task graph scheduling example

In this example, the root task Ta is always scheduled first. Next, Tc has a small weight and is highly dependent on Ta, so it is scheduled on the same processors. Next, Tb has a higher weight and small dependency and so it can be scheduled on another processor, finally Td remains and can go to any of the two processors but since the total weight of Ta+Tc is smaller than that of Tb it joins Tb on the same processor.

2.6 Lessons Learned

We summarize our remarks which form the basis of our motivation:

- In scheduling and load balancing, the most popular algorithms are : ROUND-ROBIN (GPFS, Hadoop, Gluster, Lustre, PVFS2) efficient in terms of network balance, but does not guarantee load balance. RANDOM (Gluster Lustre) is efficient in terms of processing overhead, but does not guarantee network nor load balance. WEIGHTED (Lustre) is efficient in terms of load balance, but requires extra processing.
- Even if high-throughtput systems have long effectively used striping data across I/O nodes in round-robin fashion, has been in place for quite some time. Workload studies on a number of platforms have shown that noncontiguous accesses are a common occurrence in parallel I/O workloads that the reduction of operations outweighs the added data transfer for a large percentage of accesses. This

technique can in fact also benefit systems with high latency networks as well in that it reduces the number of requests, for which there is often significant startup time.

- In the work [27] done on CIFS; it has been shown that there is an increase in the proportion of large files, mainly multimedia files; both studies conclude that a variable block size is preferrable and that small files, which still constitutes the majority of the file system space, should be colocated on a single larger block. The former study also found that the deeper is the namespace the smaller the file size.

Client applications have much more opportunities to give richer metadata information about file access patterns and requests, especially for related files, via higher level interfaces. This information will certainly help redistribute the data in a way that it facilitates both server workload balance, and access patterns.

3 Computation Model

3.1 Cluster Model

At first, we have to define the two metrics that affect the availability of a storage node: space and workload. The Space metric, denoted S, indicates the amount of disk space available at a node, and is calculated by summing the sizes of all files.

$$Si = \sum_{k=1}^{n} size(f_k) \tag{1}$$

The Load metric, denoted L, indicates the amount of workload required for serving a file. The heuristics for choosing a load metric are depending on the environment, we assume that for our simulation case the load need not to faithfully represent all of the performance affecting factors but should be heuristics that roughly but correctly translate the actual situations. From literature review in chapter 1, the load is linearly dependent with the amount of data transferred and the number of requests, while all the smaller constant factors such as network delay, concurrency cost..etc, can be gathered in a single variable that we call the Load Unit. Hence, the measure for load is based on a multiple of load unit and this is assumed to be sufficient for our calculations. THe laod metric is then calculated by summing for all files in a node, the number of requests for a file multiplied by its size, multipliled by the load unit LU. We write:

$$Li = \sum_{k=1}^{n} rcount(f_k) size(f_k) LU \tag{2}$$

Assuming there are n files in a node i of the cluster. Note that these metrics are heuristics based on the literature[**?**], and that these values are static and considered accurate enough to support our model.

3.2 Request Graph Model

We define a request graph as a directed acyclic graph *G(F,R)* such that Vertices (F) represent request files and Edges (R) represent a request relation between files.

The weight of a vertex (f) is defined as a function of its attributes. For now we only consider two attributes: The size attribute As and the request frequency Al, but this model can be easily extended to more attributes and metrics. To equally represent both attributes, the weight of a file is calculated as the product of its attributes. The

weight is deterministic in load balancing decisions and has an effect on both the space and load metrics, more over, each attribute can affect a metric more than the other: The size attribute directly affects the space metric and the frequency attribute directly affects the load mertric, though both are connected.

$$W(f) = AsAl \tag{3}$$

Where As is the file size and Al is the request frequency, i.e., the number of times a file is requested. Hence, we can redefine the cluster space and load metrics for node i, as follow:

$$Si = \sum_{k=1}^{n} As_k \tag{4}$$

$$Li = \sum_{k=1}^{n} W(f_k)AU \tag{5}$$

We also define the weight of a relation *R(i, j)* as the probability that a file f_i will request a file f_j. We denote:

$$R(i, j) = P(j|i) \tag{6}$$

$P(j|i)$ is a cumulative probability such that the sum of all request probabilities of files n related to f_i is: $1 = \sum_{j=1}^{n} P(j|i)$.
We assume that G has a single entry file that we call the root file, and we place no other limitations to the graph other than that all vertices are connected.

Table 2 describes a simple webgraph that fits to this model and of which sample values of attributes, weights and relations are set. Note that if the whole graph is assigned to a single node, the space and load metrics will be 15 and 30 respectively. Fig.2a shows the corresponding request graph, where weights (between brackets) and relation values (on the lines) are shown. The following section is based on it.

3.3 Placement Scenarios

Let's consider the graph shown in Figure 4, where three placement scenarios A1, A2 and A3 are considered. The squares represent nodes 1, 2 and 3 to which the files of the graph are to be assigned. The ellipses emphasize the relations that are maintained if two related files are assigned to the same node. The goal of a placement scenario is to 1) have balanced workloads on each node (sum of file weights) and 2) maintain as

表 2: Sample WebGraph Structure.

id	File	Size	Freq.	Weight	Relation(i,j)
0	index	16K	10	10	(0,1)=0.1 , (0,2)=0.3 , (0,5)=0.5 , (0,8)=0.1
1	video	100M	1	5	.
2	img	5M	3	3	(2,3)=0.7 , (2,4)=0.3
3	img1	2M	1	2	.
4	img2	1M	1	1	.
5	html	16K	5	5	(5,6)=0.6 , (5,7)=0.2 , (5,8)=0.2
6	file1	4K	3	3	.
7	file2	4K	2	2	.
8	info	1	1	1	.

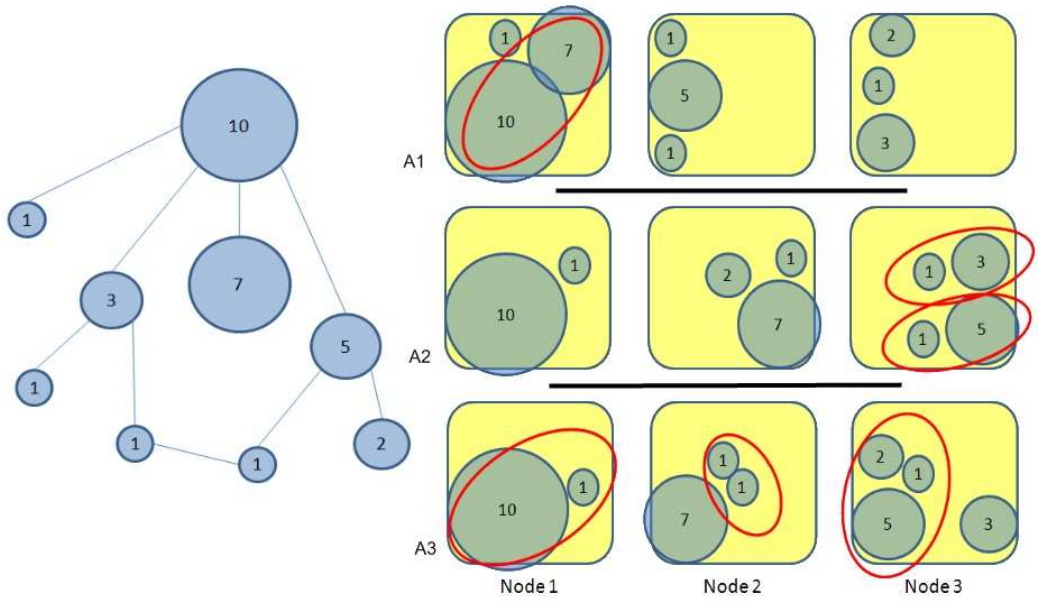

図 4: A sample request graph with different placement scenarios

may relations (as many ellipses) as possible.

Scenario A1 is a classic round robin with a left-to-right Breadth First Search (BFS). At first, *index* is assigned to node A, then its children (*mail, doc, img* and *video*) are assigned to node B, C, A and B. Then on the second pass, children of *doc* and *img* are assigned, such that *html1, html2* and *html3* for nodes C, A and B, then *img1* and

img2 for node C and A. The load metric for each node is is such that La=16, Lb=7 and Lc=10 which is quite unstable. Moreover, only 2 relations are maintained (see the ellipses), which are : (*index,img*) on A and (*doc, html*) on C.

Scenario A2 represents a weighted BFS combined with a weighted roundrobin, that we call bestFit approach. It gives priority to files with higher weights and searches for the least loaded node otherwise roundrobin when nodes are equal. At first *index* is assigned to node A, then *doc* and *video* to B and C respectively. Then, *img* to B (or C) and *mail* to C. At this level the cluster is such that (La=10, Lb=8, Lc=6). Next C is receiving *html1*, and B gets *html2* then C gets *html3*. Now it's (La=10, Lb=10, Lc=10). At the end, A receives *img1* and B receives *img2*, so that the assigned workload is La=12, Lb=11, Lc=10; which is much more balanced than classical roundrobin but still doesn't faithfully maintain file relations; again only two relations are maintained on node B only : (*doc, html*) and (*img, img1*)

Finally, scenario A3 is our ideal case where workload is nearly balanced with maximum file relations preserved, this represents the target of our work and is subsequently called BLuRGER. At first, *index* is assigned to A. Then, *Doc* is assigned because it has the highest weight and relation, it goes to B. Then *img* is assigned to B, because its weight-to-relation product is better than *video*. Next *video* goes to C. Next mail is attrated by *index* towards node A, although B and C are better. Likewise, *html1* goes to B naturally, but *html2* and *html3* are attrated to B even if C is a better choice. The same happens for *img1* and *img2* that go to C. Here, we notice that the cluster balance is as follow: La=11, Lb=11 and Lc=11; moreover, most relations are preserved, only *video* was isolated but that's acceptable because of its lower weight. The benefits hence are two-fold for both the cluster's balance, and the client's overhead.

3.4 The Placement Process

3.4.1 Approach

The placement process is a combination of two goals that are sometimes conflicting: the first goal is to maintain a good cluster balance for both space and load metrics (which is itself a challenging goal), and this requires placing heavy files on the least-loaded node. While the second goal is to minimize request graph partitioning by placing related files on the same node. To satisfy both constraints a combination of

two computation models is required, so that each model satisfy a constraint. For the first goal, a rating model is proposed to search for the least-loaded node, the rating strenght is amplified by the 'heaviness' of a file. For the second goal, an advice model is proposed to attract files which are strongly related to be placed on the same node.

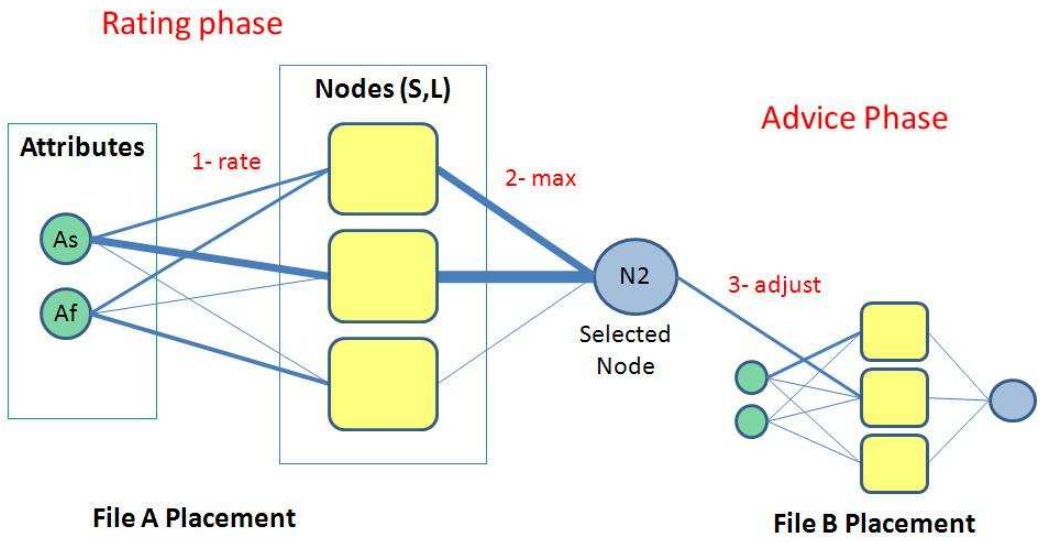

図 5: The placement process

Figure 5 shows the whole placement process that occurs in two steps. In the first step, the rating model is used to select a node to which the file will be assigned. The rating steps works in a way similar to a map-reduce function. In the map phase, or the actual rating phase, each attribute of the file (As and Al) is rated against the corresponding cluster node metric (Si and Li). On the reduce phase, the node with the maximum rating is selected for assignment. All the resource requirements/costs of the file are taken into consideration via rating and mixed in order to find a node that best satisfies all requirements by picking the node with the maximum rating. The selected node for the parent file is used by the advice model in the second step to adjust the rate of the child file. Both steps are combined to satisfy the current file's resource requirements and it's parent relation requirements. More details about each step's model are given in the following subsections.

3.4.2 Rating Model

The principle that lies behind the rating model is simple. Each file request has resource requirements, and each node has the resources that may or may not satisfy those requirements. The rating model, as the name implies, is the process by which the file rates each node with respect to the required resources. The most appropriate node is obviously the one that has the best rating. This process happens implicitly in classical weighted algorithms, but it is too simplistic, since the best node is chosen for the file, and in more sophisticated cases, the files are sorted with their weight priority. Our added value is two fold: first, there are two dimensions : the attributes dimension and the metrics dimension, hence more granularity and precision; from which the second advantage that the heavy naturally select the least loaded node without required sorting.

As for granularity, each attribute has an affinity to one or more cluster metrics. For instance, the size attribute has more impact on the space metric than the load metric, while the frequency attribute has an impact on the load and not the space. Other attributes may have affinities to one or more metrics at once. Hence, the rating for a given node i is the sum of is sub-rates, which are the rates of each attribute with regard to the corresponding metric(s), we denote:

$$rate_i(file) = \sum_{k=1}^{n} rate_i(k) \tag{7}$$

Now, let's define the rate of each of the attributes (size, frequency) in our mode with respect to the corresponding metrics (load and space), calculated as follows:

$$rate_i(As) = As\frac{(Smax - Si)}{Smax} \quad and \quad rate_i(Al) = Al\frac{(Lmax - Li)}{Lmax} \tag{8}$$

The rate is a product of the file attribute and the node metric after the metric is normalized as a fraction of the cluster-wide metric maximum, which we call appreciation. This heuristic approach is chosen so that : first, the rating mechanism is relative to the least available node of the cluster (metric max); consequently, the least available node is least likely to be chosen (appreciation = 0). Second, the appreciation is amplified by the attribute value, hence the heavier attribute are more sensitive to appreciations. Note that the appreciation is always between 0 and 1.

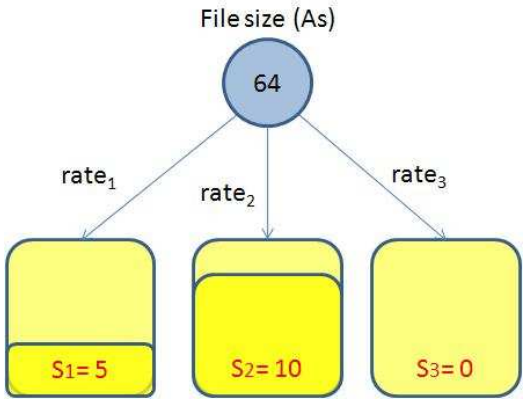

図 6: An illustration of the Rating model

Figure 6 shows an example of a file with attribute size $As = 64$ rating three nodes with different space metrics $S_1 = 5$, $S_2 = 10$ and $S_3 = 0$ respectively. Calculating the appreciation give the values of 0.5, 0 and 1.0 respectively, which means that node 1 has 50% more free space the most loaded, while node 3 has 100% free space. After multiplying the appreciation with the file attribute, the rate of each node is 32, 0 and 64 respectively. Hence, the chosen node for placement is node 3.

3.4.3 Advice Model

Now that the current file knows to which node it is assigned, it can advise its neighbors via the advice model. Indeed, the goal of the advice model is to allow two related files to be placed on the same node, especially when there are several nodes with similar availability condition. The extent to which the advice can affect placement decision depends on a priority, i.e., the weight of the requesting file (parent); on the strength of its relation to the child, and on the advice factor which is a constant to magnify furthermore the advice effect.

We denote the equation for the advice below:

$$Advice_{(i,j)} = \alpha W(f_i) R(i,j) \tag{9}$$

($Advice_{(i,j)}$) denotes the weight of the advice sent from the parent file i to the child j. The actual advice variable has two parts: the id and the weight. The id is that of the node to which the advice will lead, and the weight is the strenght of the advice. The advice weight is added to the rate of the node with the same id, and that will

affect the placement decision. The more important the parent and/or the higher the probability that it requires the child, the more it affects the selection process of the child. However, if the child weight is heavy such that it dislikes the advised node, it can counter the effect of the advice. The advice factor α is either an amplifier or a de-amplifier of the effect of the advice.

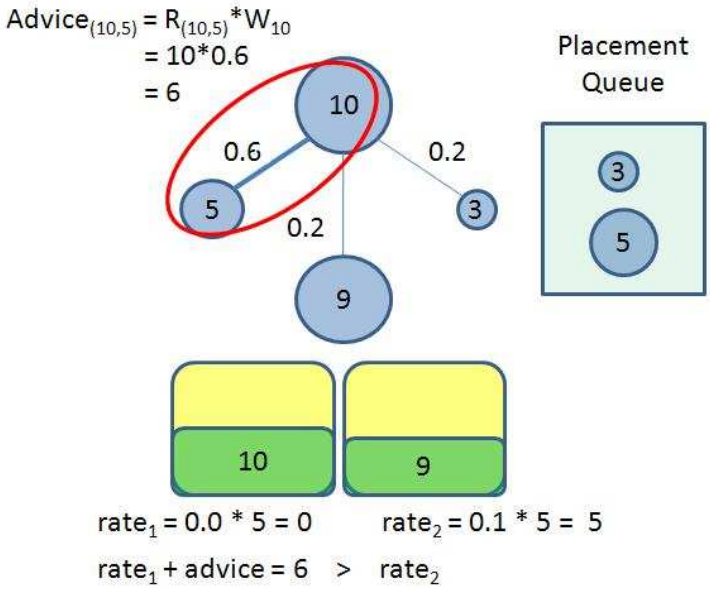

図 7: An illustration of the advice model

An example of the advice mechanism is shown in Figure 7, in which a parent with three children are to be placed on two nodes. The parent has a size attribute As=10 and the children have As values of 5, 9 and 3 respectively. The root is first placed on the left storage node and then the children are popped using BFS onto the scheduling queue. Next, the file with attribute As=9 is placed on the right node. Now, the left and right nodes have a space metric of 10 and 9 respectively. Note that the right node has only 10% free space than the left one. Now goes the turn for the file with As = 5, the rating phase gives the 0 and 0.5 respectively. However, this file has the highest relation with the parent (0.6), and when the advice is calculated and added to the rate of the left node, the latter becomes more attractive than the right and hence receives the file. Note that if the file size of 10, it will resist the advice.

3.5 Placement Algorithm

The complete placement algorithm is shown in Listing 8 below. Note that the time complexity is $O(A * N)$, as it is relevant to both the number of attributes A and the number of nodes N, and that is the only loop present at the rating phase. The advice calculation is straightforward.

Placement procedure

```
Place(file, advice)
//assigns a 'file' to a node
//using an 'advice' weight,
//returns the id of the selected node
 node = assign(file, advice
//build the next advice
 advice2.id = node.getId;
 advice2.w = file.getWeight;
//fetch edges related to 'file',
//sort by priority and push to job list
 links = list file edges
 sort(links)
 for link in links
   push link in jobs list
//recursive call for the neighbors of 'file'
 for job in jobs
   Place(job, advice2)
```

Assignment procedure

```
Assign(file, advice)
//skip file if already assigned,
//return the node assigned to it
 if(file processed)
  return file.getLocation();
//calculate node rating parameters
 calculate As, Al
 calculate Smax, Lmax
//weight each node based on above
 for Ni in nodes N
   N[i].w = rate(s) + rate(l)
//adjust the weight of the adviced node
//using advice weight and power factor
 N[advice.id] +=  factor * advice.w;
//return the node with the best rating,
//i.e., with the heaviest weigth
 return max(N)
```

図 8: Pseudo-code for core processing elements

4 Implementation

A cluster simulator was implemented to simulate client requests, the master server's handling of requests, placement and the I/O nodes status report. Java was the adopted programming language, for its elegant paradigm and extensive library which let the programmer focus on the program logic. Figure ? shows the steps of the simulation. At first, request graphs are generated using the Generator Utility, and saved as object files. This phase simulates the Consumer Client reserving the request graph on the server. Second, the Kernel program that simulates the master server, loads the graph and initiates the Cluster object which simulates the cluster nodes. The Kernel traverses the graph after reading parameters settings (e.g.: the placement algorithm), makes placement decision and updates the Cluster metrics value at each file insertion into the metadata database. After finishing placement, the master re-initiates the cluster metrics. Finally, a Client Simulator program reads the request graph from the object file and launches client threads, each thread issues a request to the master, the master updates the node metrics

4.1 Request Graph Generation

4.1.1 Anatomy of a Request Graph

The RFileSet represents the Request Graph. A graph can be implemented using either an adjacency matrix or an adjacency list. The adjacency matrix is a 2D array of size $N * N$ where N is the graph size or the number of nodes. The value at [i][j] represents the weight of the edge between node i and node j and vice versa; if the value is zero or null it means there is no edge. The adjacency list is simply a linked list of the edges to which the node is connected.

The adjacency matrix is more storage efficient for densely connected graph while the adjacency list is more efficient for quickly retrieving the edges of a given file. For this reason the adjacency list approach is chosen. Moreover, large size graphs are considered which encumbers processing for the adjacency matrix.

As shown in Figure 9, the *RFileSet* objects represents the request graph. It has a single node object which is the root node, and maintains internal list data structures for traversal. It supports operations for different traversal methods. The *RFile* is a

graph vertex object that represents the request file, it has a *RMeta* object that holds file attributes, and a list of *Relation* objects, which is the adjacency list of edges connected to it. The *RMeta* is a hashmap object that stores pairs of attribute name and value. The *Relation* object stores a references to the related file, t has two dimensions: type and value, where the 'type' is the nature of the relationships, for now; we consider the simple 'request on request' but others can be suported such as 'delete on request'; the 'value' is the probability that the relationships will actually occur.

4.1.2 Generator Utility

A graph generator (RFileSetUtils.java) can randomly generate different graph shapes (RFileSet.java) using parameters as described below:

Figure 10 shows a pseudo code of the graph generation processes. The *generate* function takes as input a *shape* variable that contains the dimension limits of the graphs, and a *rand* variable that contains random seed generators for the attributes and relations. First, the root file of the graph is built and passed to the *recursiveGenerator* function that generates children of the input parent file using the random generators, and recursively repeats the process for the children until one of the counters reaches the shape limit, and returns to the *generate* function, which completes the graph with additional The graph generator is a recursive function that starts with the root of the

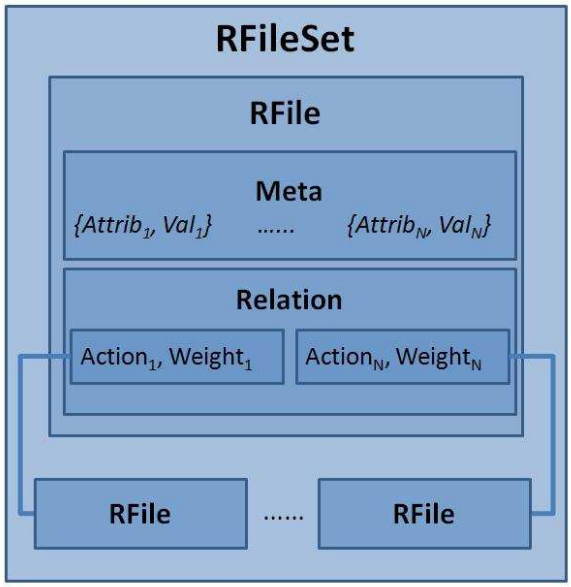

図 9: Request Graph DataStructure

generate a single graph

```
generate(shape, rand){
  init counters: size, depth, width, edges;
  RFile root = build(rand);
  recursiveGenerator(root, rand, depth+1);
  graph = new Graph().setRoot = root;
  while(counter < size )
     complete(graph)
  return graph
}

recursiveGenerator(parent, rand, depth){
  if(size > maxSize
    or width > maxWidth
    or depth > maxHeight) return
  increment size
  list children = generateNeighbors(rand, edges)
  for( link in children) do:
     make_link(parent, link, rand)
     recursiveGenerator(link, shape, rand, depth+1)
}
```

generate graph files

```
buildGraphs(){
  shapes = read params {shape1, shape2, shape3}
  rand = init randomizers(MetaGen, LinkGen)
  foreach shape in shapes
    graph = generate(shape, rand);
    write2File(graph, filename(graph+shape));
}
```

図 10: Graph Generation Pseudo-Code

表 3: Generator Parameters

Parameter	Description
maxSize	Maximum number of files in the graph
maxDepth	Maximum depth of the graph
maxWidth	Maximum number of edges per level of depth
maxEdges	Maximum number of edges allowed per node.
MetaGen	Random seed for generating attributes weights
LinkGen	Random seed for generating relation weightse

graph and keeps generating children until one of the dimension parameters (size, width, depth) is reached. If the graph size is smaller than the Size parameter, a completing function adds files until it reaches the maximum size.

The *buildGraphs* function generates many graphs with different shapes parameters initiated as a list of shapes (i.e.: dimension) and runs the single graph generator for each shape, the generate graph is saved to a file with the corresponding shape values as filename.

4.2 Cluster Simulator

The *Master* process is at the core of the simulator program because the focus is on file placement. The *metadata table* is stored on a *MySQL* Database, it shown in Table **??**. It is a simple metadata table used by the cluster simulator to store request graph files, including their attributes and relations; and to assign locations to the files after placement decisions are performed. A thread pool allows for multiple instances of client requests to arrive and safely update the table.

The Table 4 below This metadata table is queried when receiving request from the consumer client and

The *Slaves* process simulates the cluster nodes, it maintains the node workload and space status stored on a table as shown in Table 5. It is also a multithreaded process so that the same node can updates its metric concurrently in the case of multiple requests arriving at the same time.

表 4: MetaData File Table

Column	Description
ID	**Int** : A numerical value identifying the file
File	**String** : A hash of the file name
Attributes	**String** : A code that holds attribute e.g.: 1K:H means As=1 kilobyte, Al=High frequency
Relations	**String** : A code that holds relations e.g.: RonR@18873:0.5 means 50% chance to request 18873
Location	**String** : A code that contains the ID(s) of node(s) storing the file
Requests	**Int** : A request counter for this file

表 5: Slaves Table

Column	Description
ID	**Int** : A numerical value identifying the node
Hostname	**String**: identifier usable for network access
Space	**long** : The space metric
Load	**long** : The load metric
SizeMap	**String**: A String code that holds the size distribution by counting the number of files for a given node
Requests	**Int** : A request counter for this node

5 Evaluation

5.1 Strategy

The simulation strategy is divided into several steps described as follows:

1. **Graph Generation**: The graphGenerator program from Section 4 is run with a set of parameters as described in Table 6. Although several graph and cluster sizes were tested, the results shown in the following section mainly concerned different randomly generates graphs of a size of 1000, to be placed on a cluster size of 10. All of the graph attributes are randomly generated and the relations are randomly generated then adjusted with respect to the frequency attribute. The generated graphs are saved into a file, so that the same random graph is used by different programs at once. This is to simulate the early reservation step.

表 6: Experiment Parameters

Parameter	Values
Cluster Nodes	mainly 10, also 5, 15, 25, 50
Graph Size	5 sets of 1000, also 10, 100 and 10000
Graph Density	$\sqrt[2]{GraphSize}$
Attribute	random 1K, 64K, 256K, 512K, 1M, 16M, 32M, 64M, 128M, 256M
Relation Weight	random $0.01 \cdots 0.99$
Advice factor	0.0001, 0.001, 0.01, 0.1, 1, 10, 100

2. **Graph Placement**: The graph file is loaded onto the master simulator that initiates the cluster nodes and their metrics, then performs the placement in the metadata database. In addition to the proposed placement algorithm, called BLuRGER subsequently, three other algorithms are investigated: 1: RoundRobin with a Breadth First traversal of the graph, 2: Space1st which sorts the graph files by space attribute and chooses the best node with the smallest space metric; and 3: Load1st which sorts the graph files by weight and choses the best node with the smallest load metric.
3. **Client Request**: this step is executed after the previous steps are finished in

order to simulate the concept of early reservation. The simulator launches 50 client threads, each of which sends a number of requests that is relevant to the shape of the graph. At each request the master updates the Load and Space metrics. The total number of requests for each file is dependent on the frequency attribute of the file as well as the relation to its neighbors with their respective attributes. Equation 10 and Fig. 5 illustrate this concept: Al_i requests are generated for the parent, then $Al_jR(i,j)$ requests are generated for child j and $Al_kR(i,k)$ requests are generated for child k. The general formula is described in :

$$NR_i = Al_i \sum_{k=1}^{n} Al_kR(i,k) \tag{10}$$

Using this scheme, the workload on the cluster is a function of the request graph structure.

4. **Measurement**: Finally, measurements were obtained as follow: Space distribution: For each node, the sizes of files stored in that node are summed and divided by the sum of the sizes of all files in the cluster to obtain a percentage from total. For a cluster of N nodes, the closer the percentage is to $1/N$ the better the space balance. Also calculate the average standard deviation for different graphs and for different advice factors.

   Load distribution: Similar to the above, but the summation concern each file's size multiplied by the request count.

   Linking Index: count all the files that are related and that were placed together on the same node, sum the results for all nodes to obtain a percentage from the total graph.

5.2 Results

The results are comparing the following algorithms : RoundRobin, Space1st, Load1st and BLuRGER. RounRobin is a done on the files of the graph after a Breadth First Search (BFS) traversal. Space1st is a weighted placement that searches for the node with minimum space metric, after traversing the graph and sorting the files by the space attributed. Similarly, Load1st sorts the files by the weight attribute and chooses the node with the smallest Load metric. Finally, BLuRGER is the proposed method,

where unless mentionned, the used advice factor $\alpha = 0.1$. Each algorithm is run after initiating the cluster nodes.

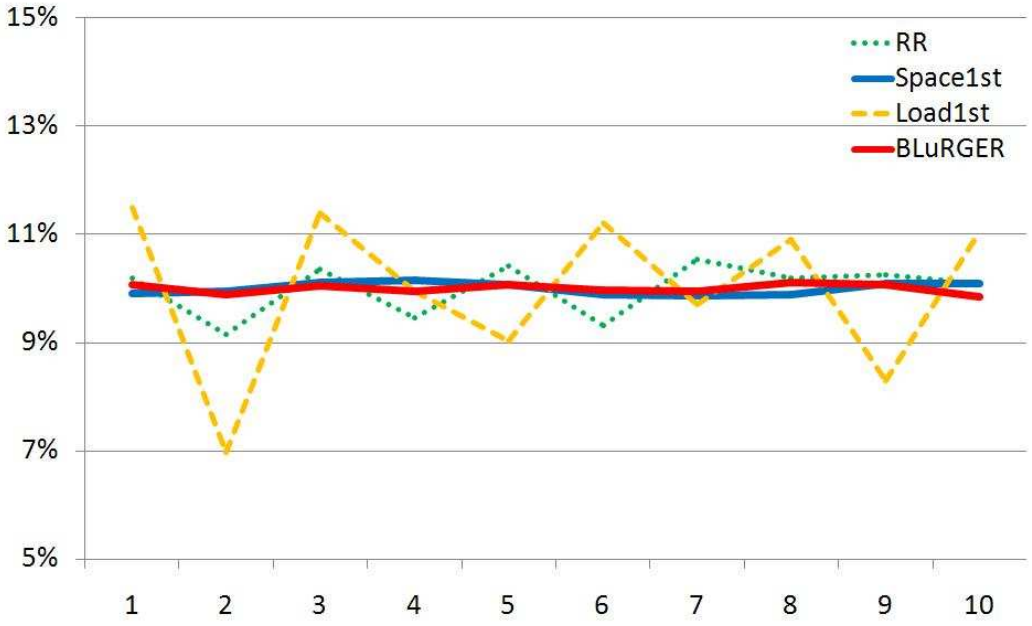

図 11: Space distribution

Figure 11 shows the distribution of used space on each node as a fraction of sum of sizes of file on the whole cluster. Since there are 10 nodes a perfectly balanced cluster will contain 10 % of the total allocated sizes. space on each node. Clearly, Space1st satisfies this requirement, and BLuRGER is performing very close to it. Obviously, RoundRobin is rather noisy but suprisingly ranks better than Load1st that tries to fit files with respect to load. Load1st is the least balanced and thus on average performs 1.59 %. behind Space1st, while RoundRobin is 1.02 %. behind. BLuRGER is very close with only 0.17 %. difference.

Figure 12 shows the distribution of load on the cluster nodes as a fraction of the total load units assigned on the whole cluster. Now Space1st is no more balanced and performs equally as bad as Round Robin with respectively 1.57 %. and 2.00 % difference from Load1st, which has a 0.25 %. variance. This time BLuRGER is not as close as in the space dimension but still outperforms the space-weighted and roundrobin approaches with less than one percent noise at 0.88 %. while it is interesting to note

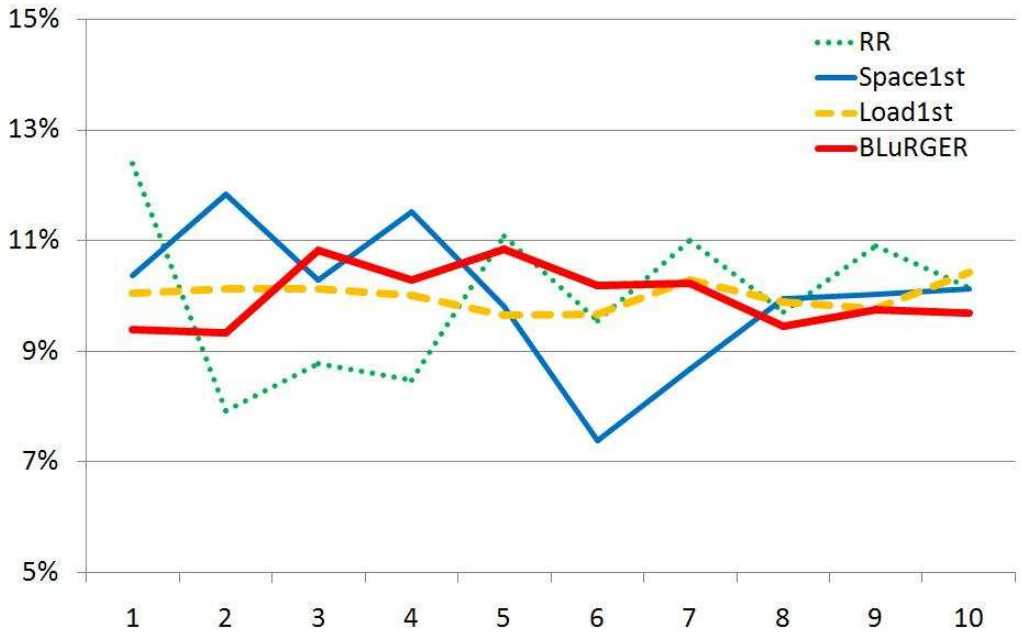

図 12: Load distribution

that the proposed method up to now combines both the advantages of space weighted and load weighted placement decisions. .

Figures 13 and 14 show the result of running the placement algorithms on five different randomly generated data sets, for the space and load metrics respectively. The goal is to see the stability of the algorithms response to different request graph shapes. The horizontal axis represents each separate dataset while the vertical axis corresponds to the standard deviation of the metric from the values for each node's metric. For the stability regarding the space metric, Figure 13....std. dev. for the total datasets...same for load

Figure 15 concerns the proposed method only. It shows the effect of the advice factor (x-axis) on the standard deviation for both the space and load metrics. The lower the standard deviation the better is the server balanced with respect to the metric. Here it appears that the balance is maintained lower than 1% for values of α lower than 1, which means that priority is given to the rating model, in which the algorithm tries to satisfy the space/load requirements first, before trying to satisfy the linking requirement. When the value of α is equal to 1 the algorithm lets the rating and advice

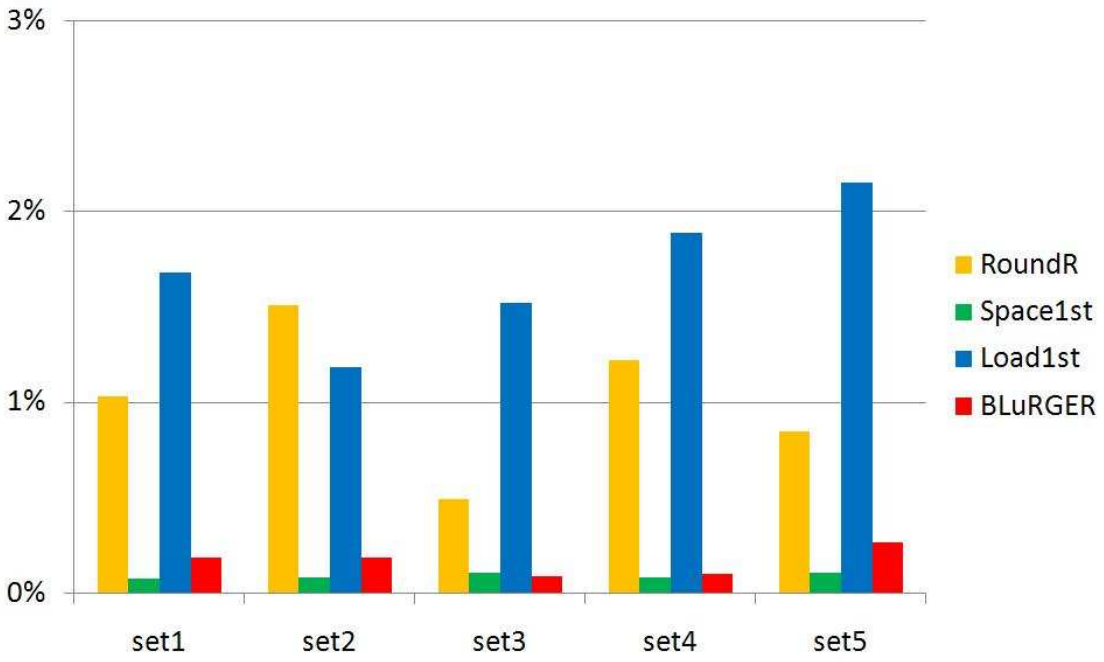

図 13: Standard deviation of Space over different datasets

models compete between themselves, in which case the files with heavy space/load requirements resist the advice, unless they are strongly bonded with the parent file and if the parent is indeed heavier. In such situation, the advising attracts neighbors to nodes with are less well rated with respect to their own space/load tentations. This situation is worsened as the advice factor α gets into higher orders of magnitude giving the power to small parent files to attract heavy children into nodes which are less and less optimal. Hence the magnitude of the cluster imbalance that lineraly grows with the magnitude of the advice factor, hinting that the advice should be used with moderation.

However, the advice factor does make a difference compared to RR, Load1st and Space1st, which is the linking index being shown in Figure 16. The linking index being the percent of related file graphs that are placed together on the same node of the cluster. The higher the linking index the less partitioned is the graph, which means less request overhead for both clients and servers. Step1 linking index considers the direct neighbors, step2 corresponds to files linked with a neighbor in addition to a neighbor of the neighbor, and step3 requires that three neighbors away be all together on the same node. Clearly, one of the strength of the proposed method can be seen in

this graph, where BLuRGER radically outperforms with more than 50% of the graph shape maintained while the others are just above 10%. Moreover, for the 2-steps the difference is of one order of magnitude; while for 3-steps all other algorithms are zeroed (besides Load1st with less than 1%) while BLuRGER still maintains a good 5%.

Figure 17 shows the evolution of the 1-step, 2-steps and 3-steps linking indexes with respect to the advice factor. They all grow linearly as the advice factor grows exponentially, which is an expected behavior.

Figure 18 shows the processing overhead associated with the an increasing graph volume. BLuRGER only slows lineraly up to 2.5% from RoundRobin and hence the delay is not a handicap especially for the case of early reservation. The processing cost overhead is actually not due to the graph size, since our algorithm traverses the graph once, however, the rating method which loops over the attributes and the cluster nodes is like to exponentially increase with those parameters. This is left to further investigation.

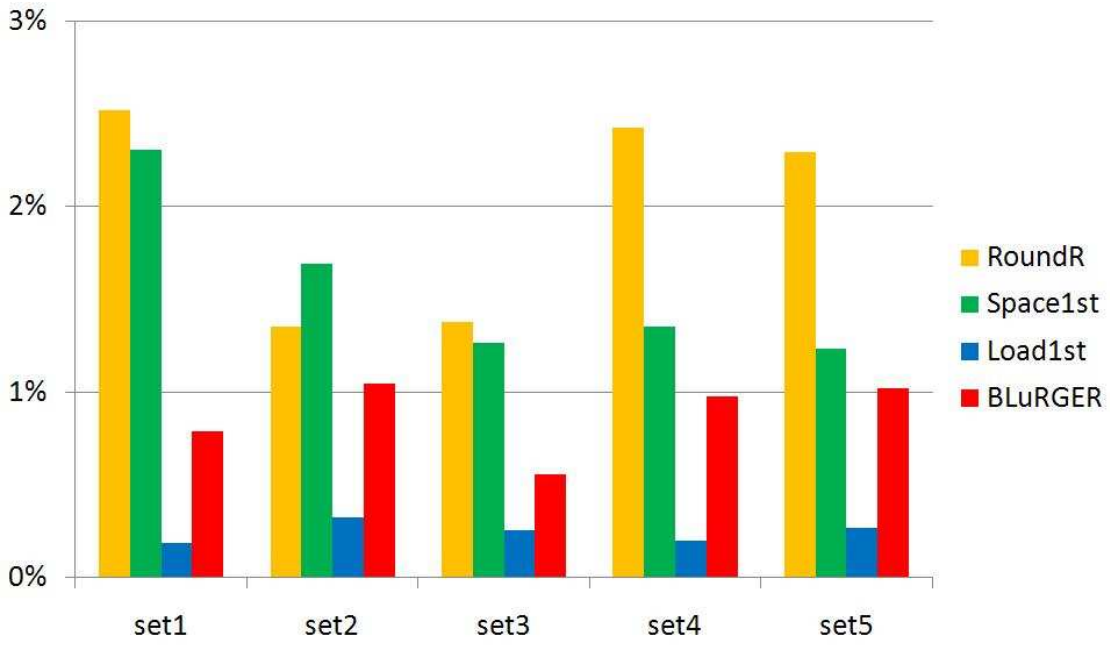

図 14: Standard deviation of Load over different datasets

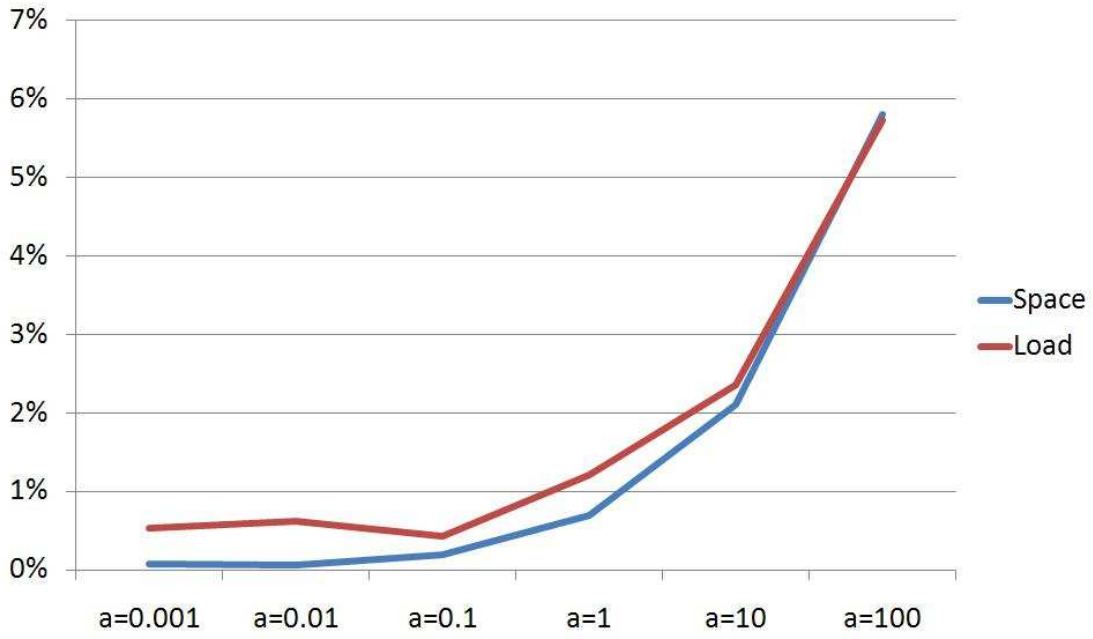

図 15: Effect of advice factor on the Space and Load distributions

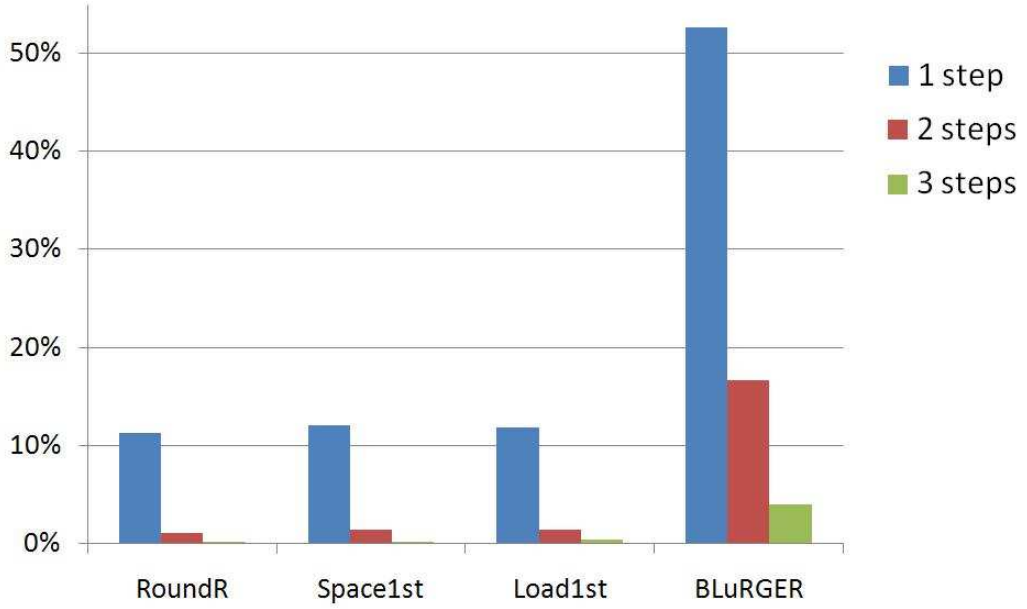

図 16: Linking index

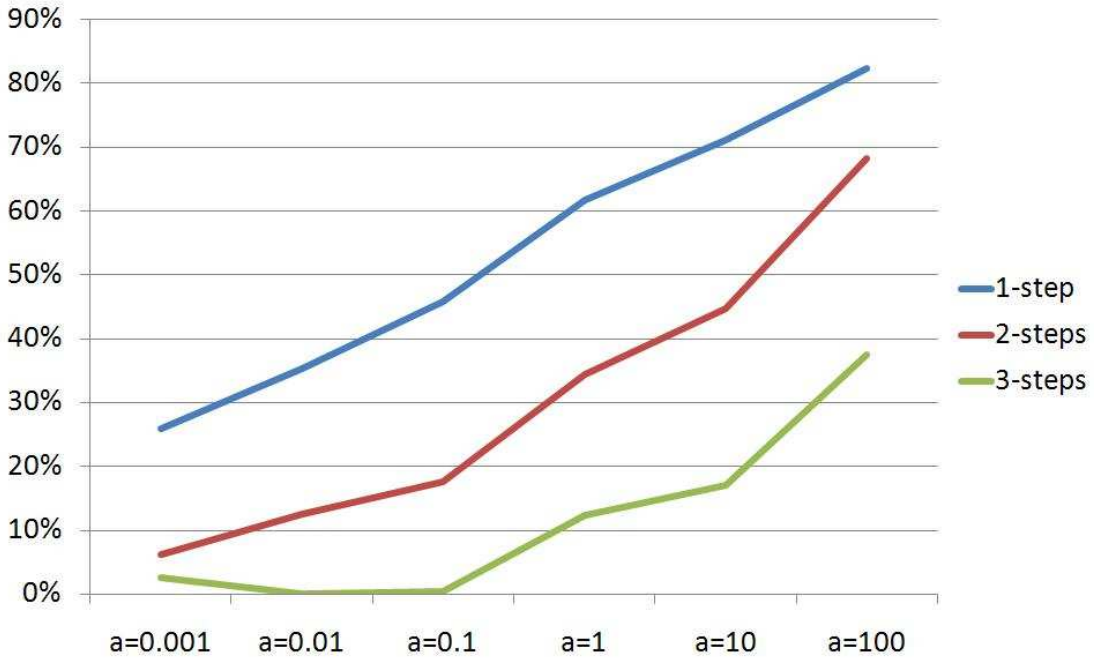

図 17: Effect of advice factor on the linking index

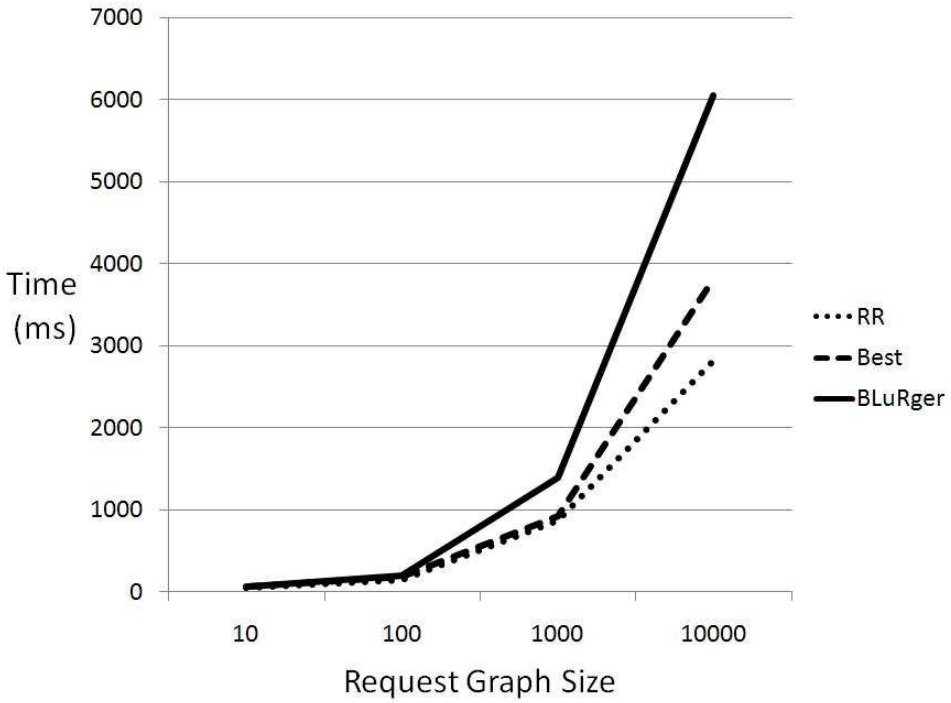

図 18: Processing overhead

6 Conclusion

6.1 Self Assessment

In this paper we have proposed a novel approach to file placement and load balancing for both clients and servers in a cluster-based file system. The approach assumes that the client knows in advance the attributes and relations of a set of files, which correspond to a request pattern. The set of connected file request was modelled as a directed acyclic graph, with file requests as vertices and request relations as edges. The weight of the vertices is the sum of the request attributes and the weight of the edges is the probability value. The goal of the proposed approach for file placement was two fold: first, minimize load imbalance on the cluster nodes by assigning the files on the best possible node; second, minimize the request overhead caused by graph partitioning so that related requests files are placed on the same node. A custom computation model was proposed to satisfy each goal, those were the Rating model and the Advice model, respectively; also, heuristics for measuring the effectiveness of each model were defined: Space and Load metrics for cluster balance and Linking index for graph partitioning. The rating model tries to satisfy the cluster balance goal by rating each request file's attribute against the cluster node metric, and selecting the node with the highest rate. The node chosen by the rating computation is fed into the the advice computation, so that the related requests parent can attract its children to the same node. This was accomplished by combining the weight of the parent with the weight of the relation in order to adjust the rate for the children.

Experiments were conducted to examine the behavior of the proposed method, in comparison to RoundRobin, Loast-weighted and Space-weighted placement algorithms. The results have shown that for cluster space balance, the proposed method performed similarly to the space-weighted, and much better than the other two. Also for cluster load balance, it performed near the load-weighted and still better than the other two. Hence, the proposed method combined the advantages of both weighted algorithms and outperformed RoundRobin. This is due to the granularity of the rating method. As for the graph partitioning, the proposed method succeeded in maintaining up to 50% of the total 1st degree relations, which was clearly far better than all the other algorithms. Also, graph shape was maintained for two and three degree away

relations for up to 10% and 5% respectively, while the other methods were near zero. These figures could be improved to up to 60% for first degrees without affecting much cluster imbalance, thanks to the amplification of the advice factor. However, an advice factor higher than 1 brought undesired noise.

To summarize, the proposed method could combine two conflicting requirements while performing as good as, if not better, than competing classical algorithms. The result are satisfying for a preliminary study, however not enough since the simulation was performed on a static system.

6.2 Future Directions

As for future work it can be divided into three parts:

1. **Dynamics support** : Clearly, the weakest point of the current method and associated simulation is the lack of support for dynamic behavior, which is actually closer to real world situation. First, load metrics should be measured dynamically as the cluster node's state is changing. This implies that it might be necessary to take into consideration further parameters during the placement even in early reservation mode; for instance, statistical measurement and traces of cluster nodes' workload could be included in the rating process.
2. **Simulator Upgrade**: The above step would certainly require that the cluster simulator as well as the clients simulator be improved to be as close as possible to a real world cluster file system. One way to verify it is to be able to reproduce the same behavior obseved on a real environment via the file access traces. Moreover, support for operations and consistency semantics, scalability and availability via replication..etc are all required.
3. **Extension to the model** : There is interesting prospects in investigating how far can the proposed method combine conflicting resource requirements for both attributes and relations on one side and cluster metrics on the other side. The request graph model is extensible with more attributes such as Expiry, Growth, Security..etc, as well as more request relations such as : read-on-write, delete-on-read ..etc. Moreover, extension of the cluster model is possible with addition of complex metrics.
4. **Graph Profiles** : Finally, after having a pseudo-real cluster file system environ-

ment and an extensible request graph model, it would be interesting to test the behavior of the system on real datasets, i.e. request graph shapes, that we shall call graph profiles. There are typical graph profiles under consideration. One profile is the Web-Media profile corresponds to a webgraph for multimedia purposes, it has properties such as heavy weight files and scarce requestion relations. Another profile is the Web-blog which corresponds to a webgraph for text and search purposes, here the files are smaller but frequencies are higher and especially relations are complex and numerous. Finally, a FS-Backup profile corresponds to the backup/restore scenario of a user file system, here the graph is tree-shaped and both attributes and relations include a wide range of values.

There is certain optimiism regarding the achieved and prospective goals, that is the opening of a new approach to client / server interaction, practically a middleware capable of satisfying complex and conflicting requirements from the client while maintaining acceptable load on the server.

参考文献

[1] : Practical File System Design with the Be File System, Morgan Kaufmann Publishers;, USA, 1st edition (1999).

[2] : Posix.FileSys structure, The Standard ML Basis Library. http://www.standardml.org/Basis/posi file-sys.html.

[3] : Cluster Computing White Paper, CoRR, Vol. cs.DC/0004014 (2000).

[4] Cluster-based scalable network services, New York, NY, USA, ACM, pp. 78–91 (1997).

[5] A scalable and highly available web server, Washington, DC, USA, IEEE Computer Society, p. 85 (1996).

[6] Cluster-based file replication in large-scale distributed systems, New York, NY, USA, ACM, pp. 91–102 (1992).

[7] Serverless network file systems, ACM Trans. Comput. Syst., Vol. 14, No. 1, pp. 41–79 (1996).

[8] Amazon S3 for science grids: a viable solution?, New York, NY, USA, ACM, pp. 55–64 (2008).

[9] Dynamo: amazon's highly available key-value store, SIGOPS Oper. Syst. Rev., Vol. 41, No. 6, pp. 205–220 (2007).

[10] The Google file system, New York, NY, USA, ACM, pp. 29–43 (2003).

[11] Scalability in the XFS file system, Berkeley, CA, USA, USENIX Association, pp. 1–1 (1996).

[12] : Distributed File System, Distributed File System Technology Center. http://www.microsoft.com/windowsserver2003/technologies/storage/dfs/default.mspx.

[13] Scalable performance of the Panasas parallel file system, Berkeley, CA, USA, USENIX Association, pp. 1–17 (2008).

[14] : The evolution of Coda, ACM Trans. Comput. Syst., Vol. 20, No. 2, pp. 85–124 (2002).

[15] Ceph: a scalable, high-performance distributed file system, Berkeley, CA, USA, USENIX Association, pp. 307–320 (2006).

[16] A taxonomy of scheduling in general-purpose distributed computing systems, IEEE Trans. Softw. Eng., Vol. 14, No. 2, pp. 141–154 (1988).

[17] : Data placement in widely distributed systems, Madison, WI, USA (2005). Supervisor-Livny,, Miron.

[18] Adaptive Load Sharing for Clustered Digital Library Servers, Washington, DC, USA, IEEE Computer Society, p. 235 (1998).

[19] : On the analysis of randomized load balancing schemes, New York, NY, USA, ACM, pp. 292–301 (1997).

[20] Load balancing a cluster of web servers using distributed packet rewriting, Boston, MA, USA (1999).

[21] Geographic Load Balancing for Scalable Distributed Web Systems, Washington, DC, USA, IEEE Computer Society, p. 20 (2000).

[22] : Interposed request routing for scalable network storage, ACM Trans. Comput. Syst., Vol. 20, No. 1, pp. 25–48 (2002).

[23] Efficient striping techniques for multimedia file servers, Austin, TX, USA (1998).

[24] Dynamic function placement for data-intensive cluster computing, Berkeley, CA, USA, USENIX Association, pp. 25–25 (2000).

[25] Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv., Vol. 31, No. 4, pp. 406–471 (1999).

[26] : Benchmarking the Task Graph Scheduling Algorithms, Washington, DC, USA, IEEE Computer Society, p. 531 (1998).

[27] A five-year study of file-system metadata, Trans. Storage, Vol. 3, No. 3, p. 9 (2007).