BLuRGER: Balancing Load using Request Graph Early Reservation

Khalil Honsali ,[†1] Shoichi Saito ,[†1] Tomoaki Tsumura [†1]
and Hiroshi Matsuo[†1]

This paper examines optimizations of file placement and load balancing on an asymmetric cluster-based file system by means of early reservations of a set of files to be requested by the client. A client defines a request graph for a set of files based on file attributes and relations, the structure of the graph is relevant to expected request patterns, which in turn have an effect on the load distribution over the cluster nodes. The server receives from the client reservations in advance to make early placement decisions in order to maintain a balanced cluster workload while satisfying the request graph structural properties and resource requirements. In this work, a model for the client request graph and a corresponding server decision algorithm are defined and discussed. The effect of this approach is simulated using variable parameters and compared against classical load balancing approaches.

1. Introduction

Over the past decade computer clusters have emerged as the dominant architecture in high performance computing. Though monolithic and centralized architectures have powerful hardware and performance, they are expensive, unavailable on failure and difficult to scale. On the other hand, computer clusters provide better performance-to-cost ratio, combining the power of hundreds or thousands of commodity off-the-shelf computers that can scale to Petabytes of storage. Cluster-based File Systems are one of the core software components on such architectures and many solutions are already available[3)1)8)], incremental scalability[7)] allows the cluster size to grow as needed but a high frequency of hardware failures requires replication mechanisms[12)] to guarantee availability of the whole system without corrupting response time[6)], finally high throughput is achieved via data caching mechanism[14)] and parallel file transfer[11)].

Becauses of the internet data boom, cluster file systems left the realm of scientific computing for use in service environments such as web search[8)] and/or computing clounds[5)]. 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 quality of service.

†1 Nagoya Institute of Technology

Our problem can be formulated as such: Assuming the Server can receive early information about the datasets to be stored, can it manage more efficiently its available resources?

Traditionally, the utilized approach for the server to be aware of data patterns was to perform statistical analysis of client requests on one side and of I/O nodes status report on the other side, and hence dynamically 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.

In the following section we give some back-

ground and related work to the problem, in section 3, the model for the early reservation request graph is discussed with the proposed placement algorithm, in section 4, the evaluation of the proposed method is shown and finally the conclusion and references in sections 5 and 6 respectively.

2. Background and Related Work

2.1 Background

Let's consider a typical[8] Cluster-based File System design such as that shown in Fig. 1. A metadata server (or master) is the central authority for both clients and I/O nodes. For I/O nodes (or slaves), the master handles critical metadata information about which nodes store what files, objects, blocks or part of files and their replicas; it also monitors their availability via the heartbeat mechanism, which is a message sent by each node to report on workload and disk space status.

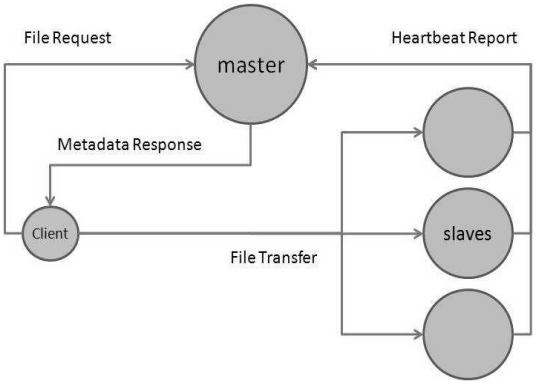

Fig. 1 Typical Cluster-based Architecture

When a client issues a request, the master looks up its metadata table and responds with information about where the file and its replicas are stored, as a metadata response. Then the client issues an I/O request directly to the corresponding slaves, and can download /upload a file's parts simultaneously from several nodes at once, which allows for increased throughput.

Load Balancing is a traditional problem in distributing computing[13]. The goal is to equally share (i.e. balance) the load between the participating nodes, so that there is no idle node when other nodes of the system are overloaded. Other connected terms are load sharing and load distributing, but they are technically more relaxed than Load Balancing. Two of the most well-known methods are random polling and round-robin. Load is a general term that refers to the amount of processing done by a computer, it can be defined as the length of CPU queue, or its average over a time period, the amount of available memory...etc The definition of the load index is essential in load balancing algorithms. We define 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. The Load metric, denoted L, indicates the amount of workload required for serving a file, including CPU processing and network operations. It is calculated by summing for all files in a node, the number of requests for a file multiplied by its size. We write:

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

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

Assuming there are n files in a node i of the cluster. Note that these metrics are heuristics based on the literature[13], and that these values are static and considered accurate enough to support our model.

2.2 Related Work

Request routing and load balancing are well studied for HTTP infrastructures[6], but the protocol messages are designed from the server perspectives. SliceFS[2] proposed the concept of a micro-proxy for routing files and requests to different servers depending on size, effectively distributing workload; but decisions were static not client-induced, and target servers were predefined, not versatile. The SEDA project[15] investigated the performance gain of adaptive scheduling using thread pools but was limited with Java Virtual Machine performance of the time. Object-based storage device (OSD)[10] is a new paradigm to storage semantics that combines the flexibility of object oriented programming with the power of file and block based storage. It is already implemented in some cluster-based file systems[4]. OSDs solve the limitation of current file system interfaces by providing higher-level semantics that can be defined as objects by the client and uploaded onto the device allowing it to understand the struc-

ture of data it stores. However, we are unaware of a load balancing scheme at the master server level that takes full advantage of the OSD paradigm. Task graph scheduling on a set of processor is a classical schedulig problem and active area of research in cluster and grid environments.A Task Graph is a set of related tasks also known as workflows. Several algorithms were developped[9] to equally distribute the load based on task priority and dependencies. Constraints specified by the client in advanced reservation[16] are taken into consideration for planning workflow distribution on the cluster. A file request is similar to a task since it requires processing and I/O resources and costs workload on the storage node, but the constraints in our problem are different and more relaxed than those of task scheduling since files do not impose a time order priority. Nevertherless, our solution is inspired from it.

3. Balancing Placement of a Request Graph

3.1 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. We define the weight of a vertex as a function of the file attributes. In this work, we are interested in attributes that affect load and space metrics, so we only consider two attributes As for space-affecting attribute and Al for load-affecting attribute, 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:

$$W(f) = AsAl \quad (3)$$

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:

$$Si = \sum_{k=1}^{n} As_k \quad (4)$$

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

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:

$$R(i,j) = P(j|i) \quad (6)$$

$P(j|i)$ 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 1 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.

Table 1 Sample Directory Structure.

File	As	Al	W	R
0 - index	1	10	10	(0,1)=0.1 (0,2)=0.3 (0,5)=0.5 (0,8)=0.1
1 - video	5	1	5	
2 - img	1	3	3	(2,3)=0.7 (2,4)=0.3
3 - img1	2	1	2	
4 - img2	1	1	1	
5 - html	1	5	5	(5,6)=0.6 (5,7)=0.2 (5,8)=0.2
6 - html1	1	3	3	
7 - html2	1	2	2	
7 - html3	1	1	1	
8 - script	1	1	1	

3.2 Placement Algorithm

Let's consider the graph shown in Fig. 2b, where three placement scenarios S1, S2 and S3 are considered. The squares represent nodes A, B and C 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 may relations (as many ellipses) as possible.

Scenario S1 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

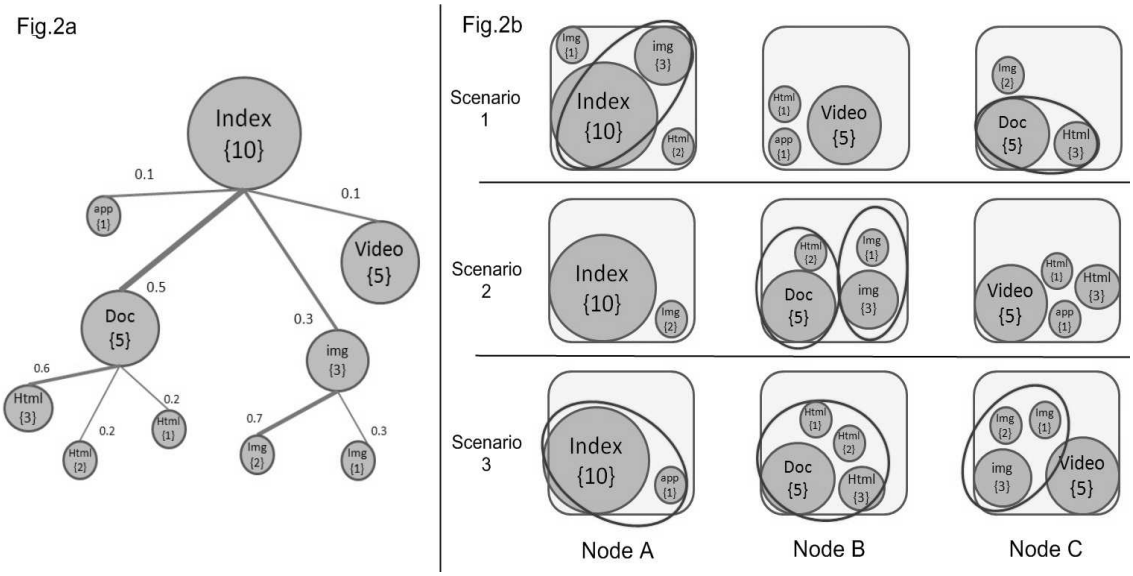

Fig. 2 a) A sample request graph. b) Different Placement methods for a)

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 S2 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 S3 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.

The listing in Fig. 3 shows the algorithm used to place the request graph onto the cluster, it starts with the root node and recursively traverses the graph until it finishes placement of all the files. It also maintains a list of processed files in order to avoid reassignment of the same file twice. The assign function is the main procedure for placement, it takes as parameters a file to assign and an advice variable, then returns the node to which the file was assigned. The node returned by the assignment function is used to build the new advice for the children (neighbors) of the current file (the parent), the weight of the advice is relative to the weight of the parent. This is a means for the currently assigned file to tell its neighbors where it has been assigned and hence 'advice' them to join him if the cluster balance allows it to. After assigning the current file, its related files (neighbors) are extracted and sorted by descending order of priority, calculated as the weight attribute times

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)
```

Fig. 3 Pseudo-code for core processing elements

the relation attribute, the sorted list is pushed to a job queue for BFS recursive traversal. This way the file requests that are most likely to follow the current file's request are assigned to the same node to reduce client overhead. If the related file weight is heavy enough it will not follow the advice of the parent and gets assigned on a different node.

Fig. 4 shows the assign function and the advice mechanism. The Assign function has two selection criteria : a criteria based on the attributes of the file and a criteria based on

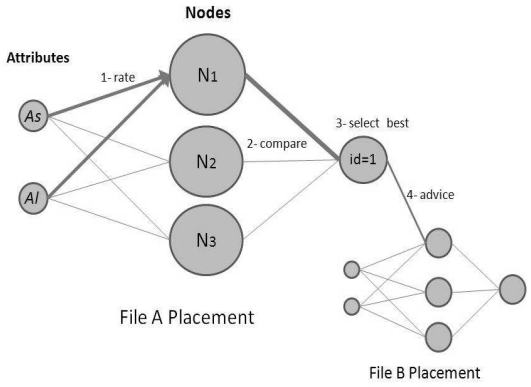

Fig. 4 Advice Model

the advice of the parent, both are combined to satisfy the current file's resource requirements and it's parent relation requirements. The attributes based selection works in a way similar to a map-reduce function. In the map phase, each attribute of the file (As and Al) is rated against a cluster node metric (Si and Li). On the reduce phase, the node with the maximum rating is selected for assignment. The rate is a product of the file attribute and the node metric, transformed to a fraction of the metric cluster maximum. This approach is chosen so that the rating mechanism is relative to the busiest node of the cluster. Consequently, the most available node (for each metric) has the highest fraction and attracts the corresponding attribute. The rating for each attribute is calculated as follows:

$$rate(s) = As\frac{(Smax - Si)}{Smax} \quad (7)$$

$$rate(l) = Al\frac{(Lmax - Li)}{Lmax} \quad (8)$$

Consider the example in Fig.4, and assume that the busiest node has both Si and Li values of 100. Also consider that (Si,Li) values for nodes 1, 2 and 3 are respectively (75, 40), (50, 100) and (75, 60). Now, for As=5 the relative rating values for each node are : 1.25 for nodes A and C and 2.5 for node B. Similarily for Al=5 attribute, the ratings are (3, 0 and 2) for nodes A, B and C respectively. When both rating functions are combined, the assignment procedure tries to satisfy all the file's requirements, that is size attribute and frequency attribute. From Fig. 4, the best node for file A is Node A with a rating of 4.25, which is better than 2.5 for B and 3.25 for C.

Now that the current file knows to which node it

is assigned, it can advise its neighbors via the advice variable. The advice variable has two parts: the id and the weight. The id is the id of the node to which the advice is sent, and the weight is the strenght of the advice. The advice weight ($Advice_{weight}$) depends on the weight of the parent (i) and the weight of its relation to the advised child (j). This is calculated as follows:

$$Advice_{weight} = W(f_i)R(i,j) \quad (9)$$

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.

4. Evaluation

4.1 Experimental Setup

We simulated a typical web-based producer/-consumer scenario, in which a producer makes reservation for a request graph, that is scheduled by the master on a 10 nodes cluster, then the consumers issue requests based on the request graph shape. Table 2 describes the different parameters used in the experiment. The 'graph size' is the number of vertices in the graph. 'graph density' is the number of edges between vertices. 'relation weight' is the probability value for each edge. 'attribute' is a discrete value for each of the vertex weight. 'advice factor' is the multiplying constant for the advice equation described above.

Table 2 Experiment Parameters

Parameter	Values
Graph Size	10, 100, 1000, 10000
Graph Density	$\sqrt[2]{GraphSize}$
Relation Weight	$0.01 \cdots 0.99$, randomly generated
Attribute	1 - 10 - 100, randomly generated
Advice factor	0.5, 1, 2

The simulation strategy is divided into three main steps described as follows: Graph Generation: A graph is generated with a set of parameters as described in table 2. In total four graph sizes with random attributes and relations are generated, then saved onto a file to be used by both step2 (the master placement) and step 3 (the consumers' request).

Graph Placement: The file graph is loaded onto the master simulator that performs the placement in the metadata database, running one of the three placement algorithms discussed in section 3.2: RoundRobin (RR), BestFit (best) and BLuRGER. For each of the three algorithms, an indepedent run of the simulator resets the metrics for the cluster nodes to initial values. In total, five runs including three runs for each advice factor just for BLuRGER.

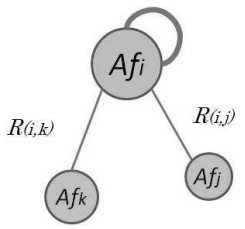

Fig. 5 Request Generation

Client Request Simulation: 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 :

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

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

4.2 Measurements and Analysis

The results are comparing the three algorithms described in Fig.2 : simple RoundRobin, BestFit and BLuRGER.

Figure 6 shows the distribution of used space as a fraction of the whole file set size. A perfect balance is logically taken by BestFit since its purpose is to look for the best space placement and hence behaves like a weighted round robin which explains the clean distribution. RoundRobin on the other side is much more unstable due to the difference in file sizes distributed sequentially. Finally BLuRGER performs in between with a comparable perfor-

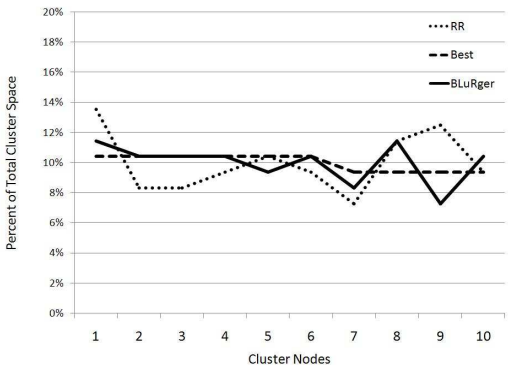

Fig. 6 Space Distribution

mance to bestFit.

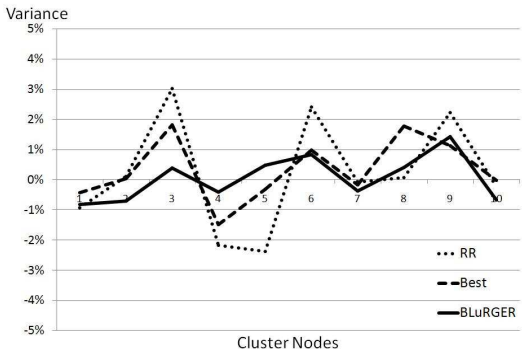

Fig. 7 Workload Distribution

Figure 7 shows the distribution of request on the cluster nodes as a variance from the mean. Although BLuRGER doesn't perfectly balance the request load because it also tries to satisfy the space metric constraint, it has yet the smallest standard deviation compared to RoundRobin and BestFit with a value of 0.01 which means that it is more stable than other methods. At the best case it can surpass RoundRobin with a 2.5% while at the worst case it is surpassed by RoundRobin by 0.8%.

Figure 8 shows the processing time of the three algorithms as the graph size scales out from 10 to 10000 nodes. RoundRobin is the fastest in processing which is logical due to the minimum assignment time, nevertheless even at the largest graph size BLuRGER processing time is only 2.14 times slower than RR which is acceptable.

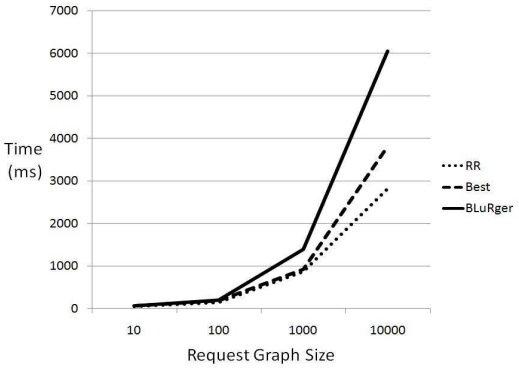

Fig. 8 Size Effect on Algorithm Processing Time

5. Conclusions and Future Work

In this paper we have proposed an original approach to client/server interactions in a cluster-based file system. By assuming that the client knows in advance the attributes and relations of a set of files, the client can make early reservation of a request to the masters in order to optimize file placement and cluster load balance. The fileset was modelled as a directed acyclic graph similar to the one used in scheduling related tasks on a set of processors. The proposed placement algorithm tries to satisfy the resource requirements for graph vertex weights and takes into considerations edges weights via an advice mechanism. The experimental data on a randomly generated graph have shown that the claims of this paper are valid compared to classical algorithms such as roundrobin or bestfit, discussed above.

As for future work it is necessary to investigate further the behavior of this algorithm on a realistic dataset such as a webgraph of considerable proportions. Moreover, the algorithm can be easily extended to support more complex attributes and relations and/or more fine grained cluster metrics. Finally, a successfull implementation will lead us to reconsider the client to server semantics and consequently the way clients store data, in order to be able to construct enough information during request graph reservations.

References

1) A.Adya, W.J. Bolosky, M.Castro, M.Cermak, R.Chaiken, J.Howell J.R.Douceur, J.R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an

incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

2) D.C. Anderson, J.S. Chase, and A.M. Vahdat. Interposed request routing for scalable network storage. *ACM Trans. Comput. Syst.*, 20(1):25–48, 2002.
3) T.E. Anderson, M.D. Dahlin, J.M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *ACM Trans. Comput. Syst.*, 14(1):41–79, 1996.
4) PhilipH. Carns, III Walter B.Ligon, RobertB. Ross, and Rajeev Thakur. Pvfs: a parallel file system for linux clusters. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference*, pages 28–28, Berkeley, CA, USA, 2000. USENIX Association.
5) Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, New York, NY, USA, 2007. ACM.
6) D. M. Dias, W. Kish, R. Mukherjee, and R.Tewari. A scalable and highly available web server. In *COMPCON '96: Proceedings of the 41st IEEE International Computer Conference*, page 85, Washington, DC, USA, 1996. IEEE Computer Society.
7) A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 78–91, New York, NY, USA, 1997. ACM.
8) S.Ghemawat, H.Gobioff, and S.T. Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM.
9) Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel Distrib. Comput.*, 59(3):381–422, 1999.
10) M. Mesnier, G.R. Ganger, and E. Riedel. Object-based storage. *Communications Magazine, IEEE*, 41(8):84–90, Aug. 2003.
11) Robert Ross, Rajeev Thakur, William Loewe, and Robert Latham. Parallel i/o in practice. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 216, New York, NY, USA, 2006. ACM.
12) H.S. Sandhu and S. Zhou. Cluster-based file replication in large-scale distributed systems. In *SIGMETRICS '92/PERFORMANCE '92: Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 91–102, New York, NY, USA, 1992. ACM.
13) Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal. Load distributing for locally distributed systems. *Computer*, 25(12):33–44, 1992.
14) Murali Vilayannur, Anand Sivasubramaniam, Mahmut Kandemir, Rajeev Thakur, and Robert Ross. Discretionary caching for i/o on clusters. *Cluster Computing*, 9(1):29–44, 2006.
15) M. Welsh, D. Culler, and E. Brewer. Seda: an architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, 2001.
16) Henan Zhao and Rizos Sakellariou. Advance reservation policies for workflows. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 47–67. Springer Verlag, 2006. Lect. Notes Comput. Sci. vol.4376.